Intel® Mobile Platform Software Development Kit

Programmer's Guide

Contents

Document Overview

This manual describes the specification and the information model used for the Adaptation subsystem of the Intel® Mobile Platform SDK.

The primary audience for this document includes the implementers and designers of mobilized software applications and those groups responsible for mobile product definitions and planning. It assumes the readership has a basic understanding of technologies used in implementing mobile applications.

Conventions Used in this Document

The following table describes the formatting conventions used in this document.

Table 1. Conventions

Formatting	Meaning
Bold Text	Denotes a term or character to be typed exactly as shown. Example: A predefined data type or function name (**HANDLE** or **uCloseConnection**), a command, or a command-line option.
Italic TextHyp	Denotes a placeholder or variable for which you must provide the actual value. Example: the statement **uCloseConnection**(*hConnection*) requires that *hConnection* be replaced with values for the *hConnection* parameters.
[ ]	Encloses optional parameters.
|	Separates an either/or choice.
…	Represents an omitted portion of a sample.
SMALL CAPITALS	Indicates the names of keys, key sequences, and key combinations. Example: ALT+SPACEBAR.
FULL CAPITALS	Indicates filenames and paths, most type structure names (which are also bold), and constants.
`Monospace`	Sets off code examples and shows syntax spacing.
Hyperlink	Indicates an active hyperlink to related information.

The References section in the appendix contains a list of useful references. The Glossary in the appendix contains a list of abbreviations and definitions.

Intel® Mobile Platform SDK Overview

This section describes the principles and goals that are guiding the architecture and design of the Intel® Mobile Platform SDK.

Architectural Principles and Goals

The Intel® Mobile Platform SDK system will be flexible and malleable. Some of the capabilities and features of the Intel® Mobile Platform SDK system are already provided by some platforms, and we expect more support for these capabilities to be added by vendors in the future. Therefore, the Intel® Mobile Platform SDK will:

- Provide a single interface as a common, or integrated, implementation that utilizes the highest layer of native platform/OS facilities offered.
- Embrace and extend existing technology and assume it will change independent of our desires or approach.
- Assume that parts of the system will be replaced.
- Be designed for fine-grained replacement, allowing the replacement of a part of the system with an external, third party implementation.

The architecture and design of the Intel® Mobile Platform SDK will allow implementation of the following:

- *Multi-architecture*—Available on the Intel® Itanium® processor family, 32-bit Intel® Architecture, Intel® Centrino™ Mobile Technology (CMT), and Intel XScale® platforms. If a feature or device is not supported on a particular platform, it has not been implemented in the library.
- *Multi-operating system*—Available on Microsoft Windows∗ XP, Microsoft Windows Server 2003, Microsoft Windows XP (64-bit), Linux*, and Microsoft Windows CE and possibly others in the future.
- *Multi-language*—Accessible from the C, C++, C#, Java*, Visual Basic*, and Common Language Runtime (CLR) languages.
- *Multi-runtime*—Useable in C Runtime, .NET* CLR, and various Java runtime environments: Java 2 Micro Edition (J2ME*), Java 2 Standard Edition* (J2SE*) and Java 2 Enterprise Edition* (J2EE*).
- *Multi-idiom*—Supports both an object-oriented and procedural call-oriented invocation idiom.

Some, but not all, of these features have been implemented in this version of the Intel® Mobile Platform SDK.

∗ Other names and brands may be claimed as the property of others.

System Architecture

Figure 1 shows the Intel® Mobile Platform SDK functional architecture.

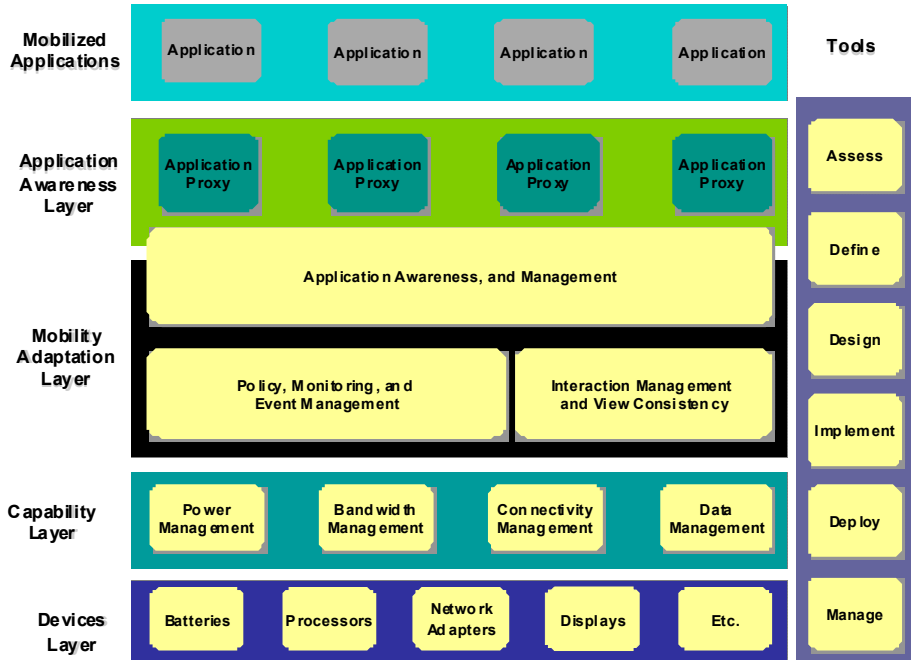

Figure 1. Intel® Mobile Platform SDK system architecture

The Intel® Mobile Platform SDK architecture has been designed to allow the following strategies:

- *Incremental implementation and value*—The system can be implemented from the bottom layer upward, while providing value at each layer.
- *Common infrastructure*—All layers use a common infrastructure and mechanisms.
- *Gradual adoption*—An application can interact with the system at any layer without requiring use of the other layers.

The layers of the Intel® Mobile Platform SDK system architecture are described below.

Programming Model

The Intel® Mobile Platform SDK uses multiple threads to achieve delivery of system events. This requires that applications that are consumers of the API be built to support a multi-threaded environment.

For the C++ binding this requires that the application be linked with the multi threaded libraries using the compiler switches /MTd or /MDd for debug builds and /MT or /MD for release builds.

For the .NET binding the application must be built to use a multithreaded apartment. This requires the use of the attribute [MTAThread] in all classes. The comsupp.lib library file will also need to be linked with the project.

If the programming model is incorrect the application will build and may apparently run but no events will be received, invalidating the operation of the Intel® Mobile Platform framework.

Device Layer

The device layer provides a common object model to represent the lower level components of the system, such as devices. It provides a mechanism to discover, enumerate, and manipulate devices independent of the platform. This layer only provides information and events.

Capabilities Layer

The capabilities layer provides the ability to discover, manipulate, and interact with system capabilities, such as power or connectivity, without knowledge of the devices that provide these capabilities. It also aggregates and virtualizes providers. For example, the power management component in the capabilities layer presents a single power view, rather than exposing batteries and other power sources. The capabilities layer only provides information and events.

Mobility Adaptation Layer

The mobility adaptation layer contains two major components: a policy component and an interaction component. These components actively respond to changes in other layers and can put into effect changes in the capabilities and devices layer to ensure that the best configuration for a given set of conditions is used.

The policy component uses user-, application-, and system-defined rules (events/conditions/actions) to determine which actions to take (actions), when to take these action (conditions), and why (events).

The interaction component provides abstract communication and invocation objects that are used by the application to interact with services and resources. The policy component can also manipulate the interaction component.

Application Awareness Layer

The application awareness layer provides a mechanism by which an application can be controlled by the policy system and provide information to lower layers about its intentions and expectations.

System Context

The Intel® Mobile Platform SDK system comprises three subsystems, as shown in Figure 2. These subsystems and their relationship to each other are described below.

Adaptation Subsystem

The Adaptation subsystem is responsible for monitoring and managing changes in the resources and context of the platform and underlying devices, such as power, connectivity, or location.

Interaction Subsystem

The Interaction subsystem is responsible for handling any interaction with local and network-based resources and services. It provides an abstract and adaptive communication architecture that hides issues, such as data and network partitioning and quality of service.

Application Awareness Subsystem

The Application Awareness subsystem provides an application "proxy" generator to generate the code required to implement a mobile application. The proxy generator uses a description of the expected behaviors and semantics of a mobile interaction to create an application specific "library" which is used by the Adaptation and Interaction subsystems to provide a transparent mobile functionality.

For example, an operation, GetCustomerInfo, could be marked as "cacheable" in the interface specification. The proxy generator would then generate code, using the Adaptation and Interaction subsystems, which would automatically cache information retrieved while online and automatically redirect the call to a cached copy of the information when offline.

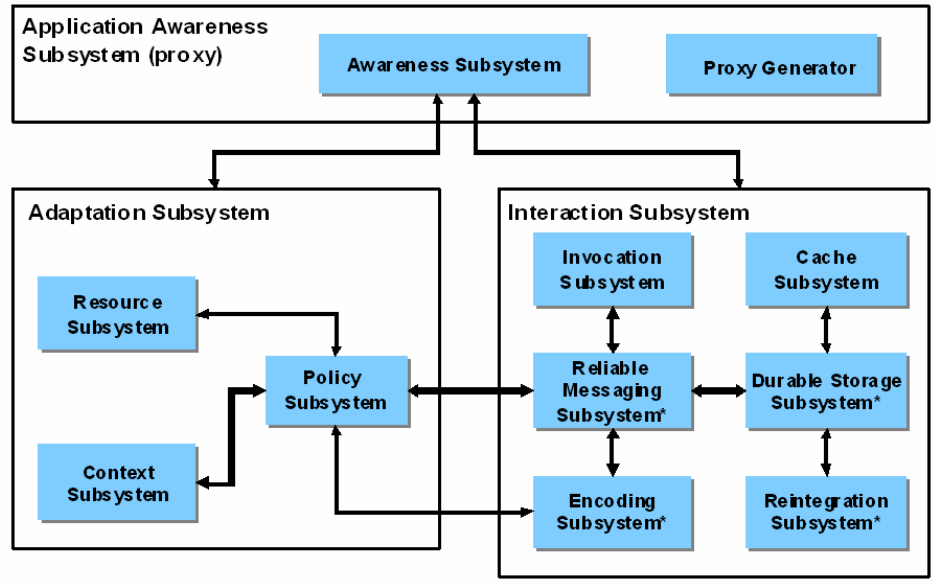

Figure 2. Intel® Mobile Platform SDK subsystems

Cross Language/Runtime

This section describes how the Intel® Mobile Platform SDK specifications are realized in the various languages and runtimes supported by the Intel® Mobile Platform SDK.

Base Data Types

The table below maps the generic base data types used in the interface specification and class definitions to corresponding data types in the target platforms.

Table 2. Data type map

Common Type	Description	C/C++	.NET CLR/CLS	Java
byte	Unsigned 8-bit	**unsigned char**	**System.Byte**	**byte**
int32	Signed 32-bit integer	**__int32**	**System.Int32**	**int**
uint32	Unsigned 32-bit integer	**unsigned __int32**	**System.UInt32**	**int**
int64	Signed 64-bit integer	**__int64**	**System.Int64**	**long**
uint64	Unsigned 64-bit integer	**unsigned __int64**	**System.UInt64**	**long**
float	32-bit floating point value	**float**	**float**	**float**
String	String	**wchar_t***	**System.String**	**java.lang.String**
Boolean	Boolean	**bool**	**System.Boolean**	**boolean**
Datetime	Date and time	**DATE**	**System.DateTime**	**java.util.Date**
Void	Untyped	**void**	**void**	**void**
Vector	Vector	**vector**	**ArrayList**	**java.util.List**
Object	Base object reference	**Object**	**System.Object**	**java.lang.Object**

***Note**: It is not clear what the advantages and disadvantages are to mapping to runtime data type "wrappers", such as **java.lang.Integer**, compared to mapping to native language data types, such as **int**. Currently, the primary types are used.*

The Java and .NET language data typing systems are each rooted from a base object: **java.lang.Object** and **System.Object**, respectively. This is not the case with C++. Therefore, the Intel® Mobile Platform SDK defines a base class, called Object, for all Intel® Mobile Platform SDK classes in the C++ binding.

For C/C++ binding, the Intel® Mobile Platform SDK String type is mapped to wchar_t* rather than being treated as a native type __wchar_t. Therefore, C/C++ applications should set the /Zc compiler option to NULL rather than wchar_t, or a link error will be generated in the application.

Constants

The table below maps the generic constants used in interface specification and class definitions to corresponding constants in the target platforms.

Table 3. Constant map

Common Constant	C/C++	.NET* CLR/CLS	Java*
True	TRUE	True	True
False	FALSE	False	False
Null	NULL	Null	Null

Scoping

The native scoping facilities of the target languages are used to define and partition a common namespace for Intel® Mobile Platform SDK interfaces, classes, etc. For C++ and .NET CLR-based languages, namespaces are used for scoping. (The term and keyword are common to both languages, but the syntax and implementation is different for each.) For Java, packages are used. The root namespaces of Intel® Mobile Platform SDK are:

C++	`Intel::Mobile`
Java	`com.intel.mobile`
.NET CLR	`Intel.Mobile`

The namespace is further partitioned into the various areas of implementation.

Interfaces

In C++, interfaces are declared as classes with pure virtual methods. In Java and .NET CLR-based languages, interfaces are declared using the native interface declarations.

Methods

Methods are defined using the common data types listed above and are implemented using the native invocation faculties of the language/runtime.

Properties

Intel® Mobile Platform SDK classes and interfaces may contain properties, such as fields or data members, with getter/setter implementations. However not all languages support properties on interfaces or implicit getter/setter. Therefore, the implicit methods Get<Property> and Set<Property> are defined for properties. For example, given a property defined in an interface as **string UserName**, an

* Other names and brands may be claimed as the property of others.

implementation includes a **SetUserName(string UserName)** method (assuming it is a settable property) and a **string GetUserName()** method.

Exceptions

Exceptions arising from Intel® Mobile Platform SDK objects and components are implemented using the native exception facilities of the languages/runtimes.

Enumerations

The native facilities of the languages/runtimes are used to define and implement enumerations. For Java, which does not support enumerations, a class with constant data members is used. The Java implementation is shown below for an enumeration called PossibleAnswers, with the following options provided for answers: DontKnow = 0, Yes = 1, No = 2 and Maybe = 3.

Java

```
public final class PossibleAnswer
{
    private static int nextValue = 0;


    public static final PossibleAnswer DontKnow = new
       PossibleAnswer();  // 0
    public static final PossibleAnswer Yes      = new
       PossibleAnswer();  // 1
    public static final PossibleAnswer No       = new
       PossibleAnswer();  // 2
    public static final PossibleAnswer Maybe    = new
       PossibleAnswer();  // 3


    private int value;


    // Constructor
    private PossibleAnswer()
    {
        this.value = nextValue;
        nextValue++;
    }


    ...
}
```

Exceptions

This section describes exceptions.

IntelMobileException

IntelMobileException is the base class for exceptions defined in the Intel® Mobile Platform SDK. For C++, this class is a new object created by the Intel® Mobile Platform SDK; For Java, it inherits from java.lang.Exception; For C#, it inherits from System.Exception.

C++

```
namespace Intel::Mobile::Base

class IntelMobileException
```

C#

```
namespace Intel.Mobile.Base

public class CIntelMobileException : Exception
```

Java

```
package com.intel.mobile.base

public class IntelMobileException extends Exception
```

Properties

Name	Type	Get	Set	Description
DetailErrorInfo	String	X		Gets the detailed error information about the current exception.
ErrorCode	String	X		Gets the error code of the current exception.
GeneralMessage	String	X		Gets a message that describes the current exception. ***Note****: for C# this property name is* **Message**.
Source	String	X		Gets or sets the name of the application or the object that causes the error.
TargetSite	String	X		Gets the method that throws the current exception.

Note*: Intel® Mobile Platform SDK relies on exceptions to report SDK errors that may be caused by operations errors, invalid interface references, or OS exceptions. These are encapsulated into IntelMobileException with the following properties:*

- **GeneralMessage**: Provides a general description of the exception. (For C#, the property **Message** is used.)
- **Source** and **TargetSite**: Provides the location where the exception occurred.
- **ErrorCode** and **DetailErrorInfo**: Provides detailed information about the error for debugging purposes.

InvalidInterfaceException

An **InvalidInterfaceException** exception is generated when an incorrect interface is used to call a method.

C++

```
namespace Intel::Mobile::Base

class InvalidInterfaceException : public IntelMobileException
```

C#

```
namespace Intel.Mobile.Base

public class CInvalidInterfaceException : CIntelMobileException
```

Java

```
package com.intel.mobile.base

public class InvalidInterfaceException extends
       IntelMobileException
```

InvalidOperationException

An **InvalidOperationException** exception is generated when a method call is invalid for the current state of the object.

C++

```
namespace Intel::Mobile::Base

class InvalidOperationException : public IntelMobileException
```

C#

```
namespace Intel.Mobile.Base

public class CInvalidOperationException : CIntelMobileException
```

Java

```
package com.intel.mobile.base

public class InvalidOperationException extends
       IntelMobileException
```

Language and Runtime Overview

Bindings

The Intel® Mobile Platform SDK adaptation subsystem provides several language and runtime binding protocols that allow applications to interact with the adaptation subsystem.

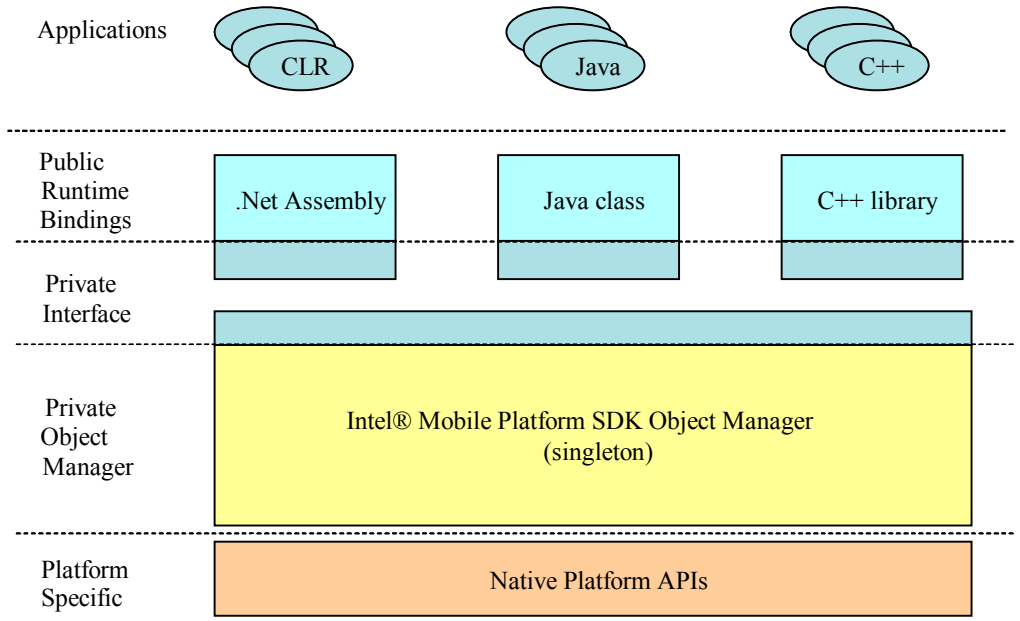

Figure 3. Runtime Binding Architecture.

Classes and Instances

Two types of objects operate within the Intel® Mobile Platform SDK adaptation subsystem:

- **Class** objects. A Class object represents a class of Instances. Class objects provide the properties, methods, and events that pertain to a group of similar entities.
- **Instance** objects. An Instance object represents "real" entities. Instance objects provide the properties, methods, and events of a particular instance.

For example, the Intel® Mobile Platform SDK adaptation subsystem contains a Class object, **BatteryClass**, which represents the concept of all batteries. The corresponding Instance object, **BatteryInstance**, represents actual batteries that exist in the system.

Note: *Instances of* ***BatteryClass****, and* ***BatteryInstance*** *are only references to the real objects which are maintained externally by the instance creator. In other words, the object lifetimes are not related; if a programmer eliminates an instance of* ***BatteryInstance****, the underlying battery is not eliminated, but the caller's reference to it is. All instances of* ***BatteryInstance*** *corresponding to a given battery, however, share the same underlying object. If a program modifies a* ***BatteryInstance*** *property using its reference, the change is reflected in all instances.*

Programmers must create an occurrence of **BatteryClass** to retrieve instances of **BatteryInstance**. To retrieve all instances, **GetInstances()** is used. To retrieve a particular instance, **GetInstance()** is used. See the following examples.

Examples

C++

```
BatteryClass myBatClass;          // The Class object for batteries
BatteryInstance * pMyBatInstance; // A Battery instance reference
// retrieve specific instance


pMyBatInstance = myBatClass.GetInstance( L"10000001" );
// the key 10000001 of the battery may be various in different
       machines.
printf( "Capacity = %d", pMyBatInstance->Capacity.GetValue() );
```

C#

```
BatteryClass myBatClass;          // The Class object for batteries
BatteryInstance myBatInstance;    // A Battery instance reference
// create a Class object
myBatClass = new BatteryClass();
// retrieve specific instance


myBatInstance = myBatClass.GetInstance( "10000001" );
// the key 10000001 of the battery may be various in different
       machines.
System.Console.Writeline( "Capacity = {0}",
       myBatInstance.Capacity.GetValue() );
```

Java

```
BatteryClass myBatClass;          // The Class object for batteries

BatteryInstance myBatInstance;    // A Battery instance reference

// create a Class object

myBatClass = new BatteryClass();

// retrieve specific instance


myBatInstance = myBatClass.getInstance( "10000001" );

// the key 10000001 of the battery may be various in different
       machines.

System.out.println( "Capacity =" +
       myBatInstance.Capacity.getValue());
```

In the preceding examples, objects for the **BatteryClass** and **BatteryInstance** are declared. A **BatteryClass** object is created and used to retrieve a specific instance. Then, the value of the Tag property is printed out.

A program can always create a Class object. An exception is created, however, if a program attempts to create an Instance object for an item that does not exist, for example, when it has been removed from the system.

Note: *Some Class objects within the Intel® Mobile Platform SDK, such as* ***BandwidthClass*** *and* ***PowerClass****, do not have Instance objects.*

Collections

The collection elements in the Intel® Mobile Platform SDK adaptation subsystem are used whenever an entity contains multiple items. For example, the Class object, **BatteryClass**, “contains” a collection of Instance objects, **BatteryInstances**. Intel® Mobile Platform SDK collections do not support adding or removing elements.

Examples

C++

```
BatteryClass myBatClass;          // The Class object for batteries

BatteryCollection *pMyBatCollection = myBatClass.GetInstances();

BatteryInstance *pMyBatInstance;

if (pMyBatCollection != NULL)

{

  // retrieve the collection of Instances

  pMyBatCollection->Reset();

  while (pMyBatCollection->HasNext())

  {

     pMyBatInstance = pMyCollection->Next();

     printf(“Capacity = %d”, pMyBatInstance->Capacity.GetValue());

     delete pMyBatInstance;

  }

  delete pMyBatCollection;

}
```

C#

```
BatteryClass myBatClass;          // The Class object for batteries

// create a Class object

myBatClass = new BatteryClass();

BatteryCollection myBatCollection =
       (BatteryCollection)myBatClass.GetInstances();

BatteryInstance myBatInstance;

if (myBatCollection!=null)

{

  myBatCollection.Reset();

  while (myBatCollection.HasNext())

  {

     myBatInstance = (BatteryInstance)myBatCollection.Next();

     System.Console.WriteLine("Capacity = {0}",
       myBatInstance.Capacity.GetValue());

  }

}
```

Java

```
BatteryClass myBatClass;          // The Class object for batteries

BatteryCollection myBatCollection;

BatteryInstance myBatInstance;

// create a Class object

myBatClass = new BatteryClass();

myBatCollection = (BatteryCollection)myBatClass.getInstances();

myBatCollection.reset();

// retrieve the collection of Instances

while (myBatCollection.hasNext())

{

  myBatInstance = (BatteryInstance)myBatCollection.next();

  System.out.println( "Capacity = " +
       myBatInstance.Capacity.getValue());

}
```

Properties

Properties are data members of **InstanceObjects**. Properties can have these characteristics:

- **Value type.** A property's value is a specific data type, such as integer, Boolean, or string.
- **Set or null.** A property can be set to a value, or not set so that it is empty (null).
- **Static or changeable.** A property can either be static or changeable. The value of a static property is set when the instance is created, and it never changes. The value of a changeable property can change after instantiation.
- **Settable or gettable.** A property is always *gettable* (its value can be retrieved). It may be *settable* by an application, but this is not always the case.

  It is possible to have a property that is changeable, but not settable. In this case, the underlying system can alter the value of the property, but an application program cannot.

The Intel® Mobile Platform SDK adaptation subsystem defines several classes that are used by Class and Instance objects to provide properties with desired characteristics.

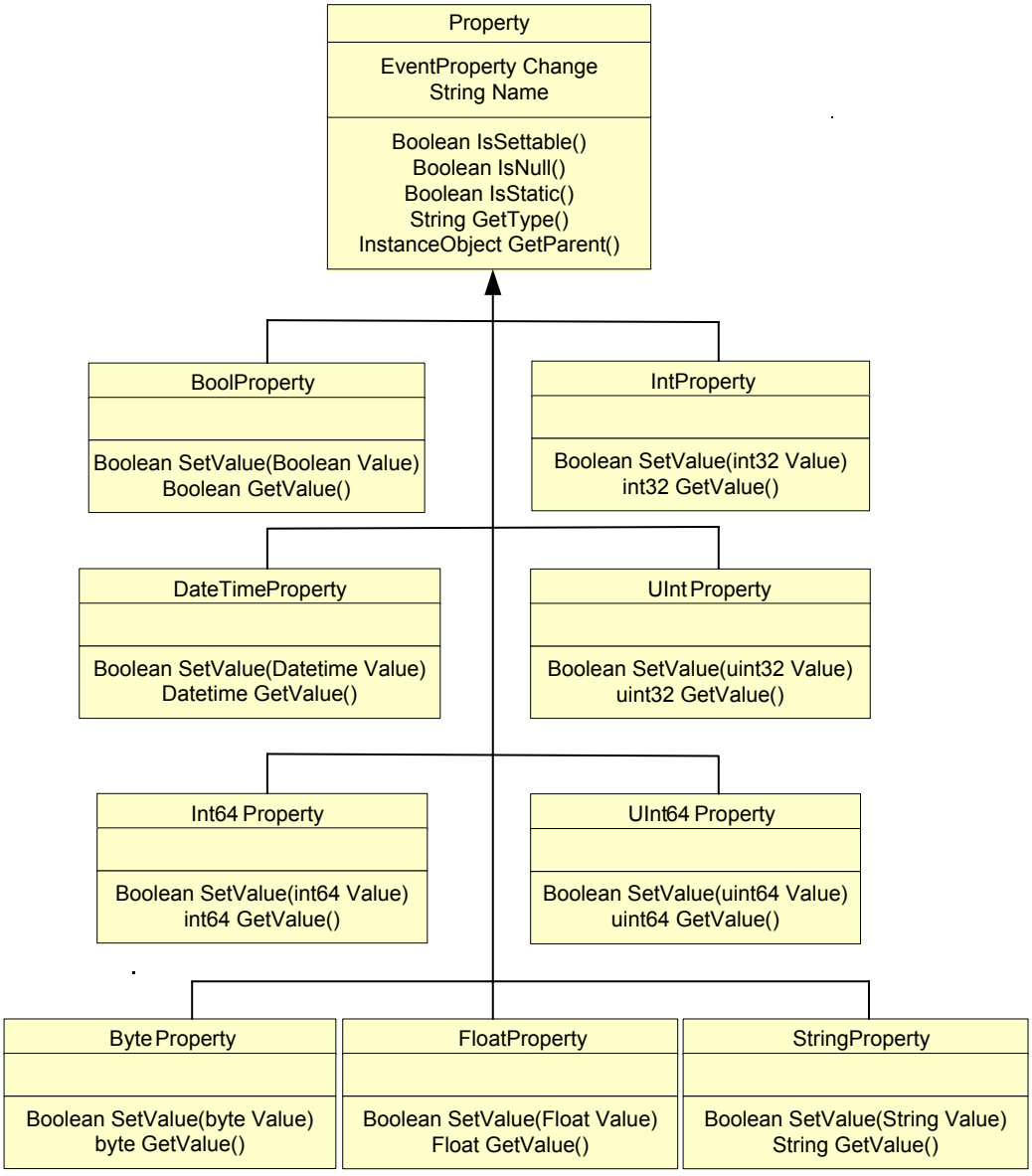

Figure 4. Properties in the Intel® Mobile Platform SDK Adaptation Subsystem.

Property Classes

The Intel® Mobile Platform SDK adaptation subsystem defines a strongly typed property class for each base data type. These are:

- **IntProperty** for integers
- **StringProperty** for strings
- **DatetimeProperty** for date/time
- **FloatProperty** for floating point data
- **…**

All property classes provide the methods **GetValue()**, **SetValue()**, **IsNull()**, **IsSettable()**, and **IsStatic()** for determining the various aspects of a property.

All property classes have an **EventProperty** named **Changed** that is issued whenever the value of the property changes. Although it is not an error to check for the **Changed** event on a static property, it is unnecessary. Applications can determine whether the property value can change using the **IsStatic()** method.

Events are properties of entities that indicate a change in the state of the entity. The Intel® Mobile Platform SDK adaptation subsystem provides standard events for Class objects, Instance objects and Properties. An **EventProperty** is used to specify the class defining each event. **EventProperties** are data members of Class objects, Instance objects and Properties, and the instance of an **EventProperty** defines an available event for the entity. Its methods **AddObserver()** and **RemoveObserver()** are used to add and remove an observer to monitor the corresponding event respectively. Instance object derivatives, such as a **NetworkAdapterInstance**, may also provide events specific to the entity. See the *Events* section for more details.

Examples

C++

```
Protocol802_11Class My802_11Class;// The Class object for 802.11

Protocol802_11Instance* pMy802_11Instance = NULL;

//A 802.11 instance reference


//retrieve specific instance

pMy802_11Instance = My802_11Class.GetInstance(L"\\?pci#ven_8...");

//the key \\?pci#ven_8... of the 802.11 may be various in
       different machines.


//get the value of the SSID

if (!pMy802_11Instance->SSID.IsNull())

{

    wchar_t* value = pMy802_11Instance->SSID.GetValue();

    wprintf( L"SSID:%s\n", value);

    delete[] value;

}

//set the value of the SSID

if (pMy802_11Instance->SSID.IsSettable())

{

    pMy802_11Instance->SSID.SetValue(L"FastAP");

}
```

C#

```
Protocol802_11Class my802_11Class = new Protocol802_11Class();

// The Class object for 802.11

Protocol802_11Instance my802_11Instance =
       (Protocol802_11Instance)my802_11Class.GetInstance("\\?pci#v
       en_8...");

//A 802.11 instance reference

//retrieve specific instance

//the key \\?pci#ven_8... of the 802.11 may be various in
       different machines.
```

```
//get the value of the SSID
if (!my802_11Instance.SSID.IsNull())
{
  string value = my802_11Instance.SSID.GetValue();
  System.Console.Writeline("SSID = {0}", value);
}
//set the value of the SSID
if (my802_11Instance.SSID.IsSettable())
{
  my802_11Instance.SSID.SetValue("FastAP");
}
```

Java

```
Protocol802_11Class my802_11Class = new Protocol802_11Class();
// The Class object for 802.11
Protocol802_11Instance my802_11Instance =
       (Protocol802_11Instance)my802_11Class.getInstance("\\?pci#v
       en_8...");
//A 802.11 instance reference
//retrieve specific instance
//the key \\?pci#ven_8... of the 802.11 may be various in
       different machines.

//get the value of the SSID
if ( !my802_11Instance.SSID.isNull() )
{
  String value = my802_11Instance.SSID.getValue();
  System.out.println("SSID = " + value);
}
//set the value of the SSID
if ( my802_11Instance.SSID.isSettable() )
{
  my802_11Instance.SSID.setValue("FastAP");
}
```

Events

Two types of events are available for ClassObject:

- **Add**: Triggered when an InstanceObject of the ClassObject is created.
- **Remove**: Triggered when an InstanceObject of the ClassObject is deleted.

For example, when a network adapter is plugged into the system, an NetworkAdapterInstance for the network adapter is created and the event **Add** of NetworkAdapterClass is triggered. When the network adapter is removed from the system, the instance NetworkAdapterInstance is deleted and the event **Remove** of NetworkAdapterClass is triggered.

One type of event is available for InstanceObject:

- **StatusChange**: Triggered when the value of any property in the InstanceObject changes.

To receive notification of an event, an application must add an observer to the event. Applications create observers by deriving a class from the **Observer** class, and then providing an implementation of the **Notify()** method. The observer is registered with an **EventProperty** using its **AddObserver()** method. When observed conditions match the target conditions, the “container” of the event calls the **Notify()** method of the observer instance.

In the examples that follow, the programmer has registered an observer that is called whenever a new network adapter instance is created, such as when a network adapter is inserted.

A **Notify()** method could be used by multiple **EventProperty**s. To distinguish events, use the **GetType()** method of the **Event** argument and compare to the desired event enumeration. This approach is shown in the following example.

Examples

C++

```
class MyObserver : public Observer
{
    virtual void Notify( const Event& event )
    {
        wstring sObjType = event.GetObjectType();
        wstring sEvent;

        switch ( event.GetType() )
        {
            case  Event::eAdd:
                sEvent = L"Add";
                break;
        }
    }
};
…
MyObserver myObserver;
NetworkAdapterClass MyNetworkAdapter;
//register the observer with the Added Event
MyNetworkAdapter.Add.AddObserver (myObserver);
```

Thresholds

Thresholds are application-defined events that indicate a certain change or transition has occurred. Thresholds are not used with static properties.

Three types of thresholds are available:

- **CounterThreshold**: Used for integer and datetime properties.
- **GaugeThreshold**: Used for integer, datetime and floating point properties.
- **ValueThreshold**: Used for string properties.

Applications are notified of threshold events using the **Notify()** method in a class derived from **Observer**. To receive notification of a threshold event, applications do the following:

- Instantiate a Threshold object
- Set the property of interest in the Threshold object.
- Specify the criteria for firing.
- Add an observer to the Threshold.

The Property to be monitored is set using the **SetObservedProperty()** method. Thresholds, like properties are strongly typed and the appropriate threshold must be used. For example, to use a counter threshold to monitor a **UIntProperty**, a threshold of type **UIntCounterThreshold** must be used. This is ensured at compile time as the arguments to the **SetObservedProperty()** method are also strongly typed.

The observer is registered with a **Threshold** using its **AddObserver()** method. When observed conditions match the target conditions, the Threshold object calls the **Notify()** method of the observer instance.

In the examples below, the programmer has registered an observer that is called whenever the charge on a battery drops below 5% or rises above 95%.

Examples

C++

```
class MyThresholdObserver : public Observer
{
   void Notify( const Event& event )
   {
      If ( event.GetType() == Event::eThresholdLow )
      {
         printf( "Battery discharged.\n" );
      }
      else If ( event.GetType() == Event::eThresholdHigh )
```

```
      {
         printf( "Battery Charged.\n" );
      }
   }
}
. . .
{
// create a Class object
BatteryClass        myBatClass;
// create an Instance object
BatteryInstance     *myBatInstance=myBatClass.GetInstance(L"10000001");
// the key 10000001 of the battery may be various in different machines.
// create an observer
MyThresholdObserver myObserver;
// create a GaugeThreshold
Int64GaugeThreshold myGaugeThreshold;

myGaugeThreshold.setLowThreshold( 5 );
myGaugeThreshold.setHighThreshold( 95 );
myGaugeThreshold.setNotify( true );
myGaugeThreshold.setObserveredProperty(
                  myBatInstance->LifePercentRemaining );
myGaugeThreshold.addObserver( myObserver );
myGaugeThreshold.start();

… continue …
```

Java

```
public class MyThresholdObserver extends Observer
{
   void Notify( Event event )
   {
      If ( event.GetType() == Event::eThresholdLow )
      {
         System.out.println( "Battery discharged" );
      }
      else If ( event.GetType() == Event::eThresholdHigh )
      {
         System.out.println( "Battery Charged" );
      }
   }
}
…
{
BatteryClass myBatClass;
MyThresholdObserver myObserver;
GaugeThreshold myGaugeThreshold;

// create a Class object
myBatClass = new BatteryClass();
// create an Instance object
BatteryInstance myBatInstance=myBatClass.getInstance("10000001");
// the key 10000001 of the battery may be various in different
      machines.
// create a GaugeThreshold
myGaugeThreshold = new GaugeThreshold();
// create an observer
myObserver = new MyThresholdObserver();

myGaugeThreshold.setLowThreshold( 5 );
myGaugeThreshold.setHighThreshold( 95 );
```

```
myGaugeThreshold.setNotify( true );
myGaugeThreshold.setObserveredProperty(
                    myBatInstance.LifePercentRemaining );
myGaugeThreshold.addObserver( myObserver );
myGaugeThreshold.start();


… continue …
```

Capabilities

For certain classes of objects, only one instance object is available. Capabilities fall into this category. For this reason, all the capabilities for a given system are grouped into a single class, the **CapabilityClass**. Three classes fall into the capability category: **Connectivity**, **Bandwidth**, and **Power**.

Like other **ClassObjects**, capabilities can be enumerated using an **InstanceCollection**. Because capability **InstanceObjects** are static, they are different from other **ClassObjects** in a few key ways:

- The collection cannot be invalidated.
- **Add** and **Remove** events on the class will never be triggered.
- When enumerating capabilities, a specific type of capability can be used by treating the InstanceObject as a specific type using casting.

Each **Capability** has its own key and type that matches the specific name of the **Capability**. For example, **GetType()** and **GetKey()** on a **ConnectivityInstance** will return “Connectivity”. This allows a programmer to gain access to a specific capability using the **GetInstance()** method of the **ClassObject** as shown in the following example. For more information on specific capabilities, see the **Connectivity**, **Bandwidth**, and **Power** sections below.

Examples

C++

```
CapabilityClass MyCapabilityClass;

CapabilityInstance* pMyCapabilityInstance;

ConnectivityInstance* pMyConnectivityInstance;


pMyCapabilityInstance = MyCapabilityClass.GetInstance(
       L"Connectivity" );

if ( pMyCapabilityInstance != NULL )

{

       pMyConnectivityInstance =
       dynamic_cast<ConnectivityInstance*>(pMyCapabilityInstance);

       if ( !pMyConnectivityInstance->Connected.IsNull() )

       {

              wprintf( L"Connected: %s\n", pMyConnectivityInstance-
       >Connected.GetValue()?L”True”:L”False” );

       }

       delete pMyCapabilityInstance;

}
```

Programmer's Reference

Core Classes

This section describes the Intel® Mobile Platform SDK adaptation subsystem core classes. Core classes are base classes used by the Intel® Mobile Platform SDK and are never instantiated directly by an application.

Properties

Property classes for various data types are used to define the properties of Instance objects.

C++

```
namespace Intel::Mobile::Base

class IntProperty : public Property

class Int64Property : public Property

class FloatProperty : public Property

class DateTimeProperty : public Property

class BoolProperty : public Property

class StringProperty : public Property

class ByteProperty : public Property

class UIntProperty : public Property

class UInt64Property : public Property
```

C#

```
namespace Intel.Mobile.Base

public class FloatProperty : Property

public class DateTimeProperty : Property

public class BoolProperty : Property

public class StringProperty : Property

public class ByteProperty : Property

public class IntProperty : Property

public class Int64Property : Property

public class UIntProperty : Property

public class UInt64Property : Property
```

Java

```
package com.intel.mobile.base

public class FloatProperty extends Property

public class DateTimeProperty extends Property

public class BoolProperty extends Property

public class StringProperty extends Property

public class ByteProperty extends Property

public class IntProperty extends Property

public class Int64Property extends Property

public class UIntProperty extends Property

public class UInt64Property extends Property
```

Properties

Name	Type	Get	Set	Description
Name	**String**	X		The property name. ***Note:*** *For the C++version, a returned string that is unused must be manually deleted to avoid a memory leak.*

Methods

Return Value	Signature	Description
type	**GetValue()**	Valid for all properties.
Boolean	**SetValue(** *type Value* **)**	Changes only valid for settable properties. Returns **true** if the property has been set successfully.
String	**GetType()**	Returns a string containing the type name of the derived class, such as “IntProperty”. ***Note:*** *For the C++version, a returned string that is unused must be manually deleted to avoid a memory leak.*
Boolean	**IsNull()**	Returns **true** when • The property is not provided by lower layers. • The property’s value has not been set, such as when its value is unknown. Valid for all properties.
Boolean	**IsSettable()**	Returns **true** if the property can be set by the caller. Valid for all properties.
Boolean	**IsStatic()**	Returns **true** if value does not change. Valid for all properties
InstanceObject	**GetParent()**	Returns the **InstanceObject** for which the property is an attribute.

Events

Name	Type	Description
Changed	**EventProperty**	Valid for non-static properties

Remarks

If a property is not set, **IsNull()** will return **true**, and the value returned by **GetValue()** will be undefined.

If a manufacturer has not implemented a property for a device, the value returned by **GetValue()** for the property will be NULL. **IsNull()** should be invoked first to determine if a property is provided on a device. For example, on an IBM-T42 system, the following properties of the battery return NULL: **CycleCount**, **ManufacturerDate**, **SerialNumber**, and **Temperature**. Due to the large number of available devices, we cannot provide a complete list of such unimplemented device properties or device types.

Property Arrays

Property Array classes for arrays of various data types are used to define the properties of Instance objects.

C++

```
namespace Intel::Mobile::Base

class FloatArrayProperty : public ArrayProperty

class DateTimeArrayProperty : public ArrayProperty

class BoolArrayProperty : public ArrayProperty

class StringArrayProperty : public ArrayProperty

class ByteArrayProperty : public ArrayProperty

class IntArrayProperty : public ArrayProperty

class Int64ArrayProperty : public ArrayProperty

class UIntArrayProperty : public ArrayProperty

class UInt64ArrayProperty : public ArrayProperty
```

C#

```
namespace Intel.Mobile.Base

public class FloatArrayProperty : ArrayProperty

public class DateTimeArrayProperty : ArrayProperty

public class BoolArrayProperty : ArrayProperty

public class StringArrayProperty : ArrayProperty

public class ByteArrayProperty : ArrayProperty

public class IntArrayProperty : ArrayProperty

public class Int64ArrayProperty : ArrayProperty

public class UIntArrayProperty : ArrayProperty

public class UInt64ArrayProperty : ArrayProperty
```

Java

```
package com.intel.mobile.base

public class FloatArrayProperty extends ArrayProperty

public class DateTimeArrayProperty extends ArrayProperty

public class BoolArrayProperty extends ArrayProperty

public class StringArrayProperty extends ArrayProperty

public class ByteArrayProperty extends ArrayProperty

public class IntArrayProperty extends ArrayProperty

public class Int64ArrayProperty extends ArrayProperty

public class UIntArrayProperty extends ArrayProperty

public class UInt64ArrayProperty extends ArrayProperty
```

Properties

Name	Type	Get	Set	Description
Name	***String***	*X*		The property name. ***Note:*** *For C++ version, the returned string needs to be deleted manually when unused to avoid memory leak.*

Methods

Return Value	Signature	Description
Type	**GetValue(uint32** *Offset***)**	Valid for all properties
Void	**SetValue(uint32** *Offset*, *type Value* **)**	Changes only valid for settable properties.
uint32	**GetSize()**	
Boolean	**IsNull()**	Returns **true** when: • The property is not provided by lower layers. • The value of the property has not been set, such as when its value is unknown. Valid for all properties.
Boolean	**IsSettable()**	Returns **true** if the property can be set by the caller. Valid for all properties.
Boolean	**IsStatic()**	Returns **true** if value does not change. Valid for all properties
InstanceObject	**GetParent()**	Returns the **InstanceObject** for which the property is an attribute.

Events

Name	Type	Description
Changed	**EventProperty**	Valid for settable properties

Remarks

If a property is not set, **IsNull()** will return **true**, and the value returned by **GetValue()** will be undefined.

If a manufacturer has not implemented a property for a device, the value returned by **GetValue()** for the property will be NULL. **IsNull()** should be invoked first to determine if a property is provided on a device. For example, on an IBM-T42 system, the following properties of the battery return NULL: **CycleCount**, **ManufacturerDate**, **SerialNumber**, and **Temperature**. Due to the large number of available devices, we cannot provide a complete list of such unimplemented device properties or device types.

ClassObject

ClassObject is the base for all Class objects.

C++

```
namespace Intel::Mobile::Base

class ClassObject : public Object
```

C#

```
namespace Intel.Mobile.Base

public class ClassObject : Object
```

Java

```
package com.intel.mobile.base

public abstract class ClassObject extends Object
```

Events

Name	Type	Description
Add	**EventProperty**	Indicates a new Instance of the Class was created.
Remove	**EventProperty**	Indicated an existing instance of the Class was deleted

Methods

Return Value	Signature	Description
InstanceCollection	**GetInstances()**	Returns a collection of Instance objects. ***Note:*** *For the C++version, a returned* **InstanceCollection** *object that is unused must be manually deleted.*
InstanceObject	**GetInstance(String** *Key* **)**	Returns the Instance object specified by the key. ***Note:*** *For the C++version, a returned* **InstanceObject** *that is unused must be manually deleted.*
String	**GetType()**	Returns a string containing the type name of the derived class, such as “NetworkAdapter”. ***Note:*** *For the C++version, a returned string that is unused must be manually deleted to avoid a memory leak.*
String	**GetObjectType()**	Returns the string “ClassObject”. ***Note:*** *For the C++version, a returned string that is unused must be manually deleted to avoid a memory leak.*

Remarks

On Windows XP unmanaged runtime, *InstanceCollection* and *InstanceObject* refer to strongly typed collections and instances. For example, in the case of batteries, they are **BatteryCollection** and **BatteryInstance**, respectively.

For other runtimes such as .NET 1.1, J2SE 1.4.2, and Microsoft Windows Mobile* 2003, that don’t support the covariant return type, *InstanceCollection* and *InstanceObject* only refer to themselves.

InstanceObject

InstanceObject is the base for all Instance objects.

C++

```
namespace Intel::Mobile::Base

class InstanceObject : public Object
```

C#

```
namespace Intel.Mobile.Base

public class InstanceObject : Object
```

Java

```
package com.intel.mobile.base

public abstract class InstanceObject extends Object
```

Events

Name	Description
StatusChange	Indicates a property value of the Instance has changed.

Methods

Return Value	Signature	Description
String	**GetType()**	Returns a string containing the type name of the derived class, such as “WiredAdapter” ***Note:*** *For the C++version, a returned string that is unused must be manually deleted to avoid a memory leak.*
String	**GetKey()**	Gets the unique identifier of the **InstanceObject**. This provides the ability to query from the **ClassObject** for a specific instance of a given class. This is only unique within a given class. The key of an InstanceObject may vary across devices. Keys are used internally and are not intended to be parsed or used externally. ***Note:*** *For the C++version, a returned string that is unused must be manually deleted to avoid a memory leak.*
String	**GetObjectType()**	Returns the string “InstanceObject”. ***Note:*** *For the C++version, a returned string that is unused must be manually deleted to avoid a memory leak.*

Remarks

Enumerators only allow reading the data in the collection. Enumerators cannot be used to modify the underlying collection.

InstanceCollection

InstanceCollection is the base for all Collection objects.

C++

```
namespace Intel::Mobile::Base

class InstanceCollection : public Object
```

C#

```
namespace Intel.Mobile.Base

public class InstanceCollection: Object
```

Java

```
package com.intel.mobile.base

public abstract class InstanceCollection extends Object
```

Methods

Return Value	Signature	Description
InstanceObject	**Current()**	Returns the current instance object in the collection. Note that **Current()** returns the same object until **Next()** or **Reset()** is called. **InvalidOperationException** is returned if the enumerator is positioned before the first element of the collection or after the last element. ***Note:*** *For the C++version, a returned* **InstanceObject** *that is unused must be manually deleted.*
Boolean	**HasNext()**	Returns **true** if the iteration has more elements. For example, a **true** is returned if **Next()** would return an element rather than throwing an exception.
Void	**Reset()**	Sets the enumerator to its initial position before the first element in the collection.
InstanceObject	**Next()**	Returns the next instance object in the collection. **InvalidOperationException** is returned if the iteration has no more elements. ***Note:*** *For the C++version, a returned* **InstanceObject** *that is unused must be manually deleted.*
String	**GetObjectType()**	Returns the string “InstanceCollection”. ***Note:*** *For the C++version, a returned string that is unused must be manually deleted to avoid a memory leak.*

Remarks

When a collection is created or after a call is made to **Reset(),** the enumerator is positioned before the first element of the collection. The first call to **Next()** moves the enumerator to the first element of the collection.

After the end of the collection is passed, subsequent calls to **Next()** return **false** until **Reset()** is called.

An enumerator remains valid as long as the collection remains unchanged. If changes are made to the collection, such as adding, modifying, or deleting elements, the enumerator is irrecoverably invalidated and the next call to **Next()**, or **Reset()** throws an exception, **InvalidOperationException**.

On Windows XP unmanaged runtime, *InstanceObject* refers to strongly typed collections and instances. For example, in the case of batteries, the instance object is **BatteryInstance**.

For other runtimes such as .NET 1.1, J2SE 1.4.2, and Windows Mobile 2003, that don’t support the covariant return type, *InstanceCollection* and *InstanceObject* only refer to themselves.

Observer

Observer specifies the abstract base class for all observers.

C++

```
namespace Intel.Mobile.Base

class Observer
```

C#

```
namespace Intel.Mobile.Base

public class Observer
```

Java

```
package com.intel.mobile.base

public abstract class Observer
```

Methods

Return Value	Signature	Description
Void	**Notify**(**Event** *event*)	A programmer-provided implementation that is called when the event is fired.
Void	**Notify(const Event&** *event*) = 0;	Base class declaration.

Remarks

Observer is the base class for event observers. To be notified of an event, the programmer must implement the following steps:

- Derive a class from **Observer.**
- Provide an implementation for the **Notify()** method.
- Use the **AddObserver()** method of one or more **EventProperty**s to register for an event.

When a registered event occurs, the **Notify()** method of the Observer specialization is called.

Thresholds also create events when conditions are met. To be notified of Threshold events, the programmer must implement the following steps:

- Create one or more **Threshold** objects.
- Derive a class from **Observer.**
- Provide an implementation for the **Notify()** method.
- Call the **AddObserver()** method of one or more of the **Threshold** objects to register for the events provided by the type of Threshold.

Calls to the **Observer Notify**() method are made on a thread initiated by the Intel® Mobile Server process and not the primary application thread. The programmer must take this into account when implementing an **Observer Notify**() method. Processing within the scope of the call to the **Notify**() method

must be kept to a minimum and data access must be performed in a thread-safe manner using synchronization objects where necessary.

The methods exposed by the Event class permit access to the object that originated the event. These methods are designed to allow the identification of the event source.

Although it is possible to create one Observer implementation that can be used for all events, considerable code may be required within the Notify method to determine which event fired and the consequent action to perform.

A more natural approach is to define one Observer per situation, grouping together events that require the same response. For example, an Observer may be defined that performs one or more actions when network connectivity changes.

When creating an **Observer** object, the programmer must ensure that the **Observer** object is valid when registered events are triggered. Before quitting the function in which the **Observer** object was created and registered, it is recommended that **RemoveObserver()** be invoked. If **RemoveObserver()** is not explicitly invoked, registered events could be triggered outside of the scope of the function after the function exits. This may lead to the Intel® Mobile Platform SDK attempting to notify an observer that no longer exists.

See Also

Event, Threshold

EventProperty

EventProperty specifies the class defining the event for a change in state of an entity.

C++

```
namespace Intel::Mobile::Base

class EventProperty
```

C#

```
namespace Intel.Mobile.Base

public class EventProperty
```

Java

```
package com.intel.mobile.base

public class EventProperty
```

Methods

Return Value	Signature	Description
Boolean	**AddObserver(Observer** *o* **)**	Adds the observer to the event. When observed conditions match the target conditions, the **Notify()** method of the observed object is invoked. Returns **true** if successful.
Boolean	**RemoveObserver(Observer** *o* **)**	Removes the observer from the event. Returns **true** if successful.

Remarks

EventProperties are data members of **ClassObject**s, **InstanceObject**s and **Properties**.

Each instance of an **EventProperty** in the system defines an available event for the entity that it is a member of. For example all Properties have attributes of type **EventProperty** named **Changed**.

For the Java version, since different JREs have different garbage collection mechanisms, **AddObserver** and **RemoveObserver** must be called in pairs to ensure all unused resources are released before exit.

See Also

Observer, Event

Event

Event defines methods that identify the type of an event, the time it occurred and the object that originated the event.

C++

```
namespace Intel::Mobile::Base

class Event : public EventBase
```

C#

```
namespace Intel.Mobile.Base

public class Event : EventBase
```

Java

```
package com.intel.mobile.base

public class Event extends EventBase
```

EventType

The table below lists values for the types of events that can occur.

Value	Meaning	Owner
eChanged	The value of the property has been changed.	Property
eAdd	An instance has been added into the system.	ClassObject
eRemove	An instance has been removed from the system.	ClassObject
eConnect	The first network adapter has connected to the network.	Connectivity Instance Object
eDisconnect	The last network adapter has disconnected from the network.	Connectivity Instance Object
eRouteTableChange	The routing table has changed.	Connectivity Instance Object
eIPAddressChange	The IP address has changed.	Connectivity Instance Object
eStatusChange	The status of an instance has changed.	InstanceObject
eMediaConnect	When media is attached to a protocol instance that previously had no media attached.	LinkProtocol Instance Object
eMediaDisconnect	When an instance of a protocol no longer has media attached	LinkProtocol Instance Object
eThresholdReached	The value of the property has reached the threshold.	CounterThreshold
eThresholdLow	The value of the property has become lower than the threshold.	GaugeThreshold

Value	Meaning	Owner
eThresholdHigh	The value of the property has become higher than the threshold.	GaugeThreshold
eThresholdEqual	The value of the property has remained equal to the threshold.	ValueThreshold
eThresholdDiffer	The value of the property has become different from the threshold.	ValueThreshold
eShutDownRequested	A request has been sent out to shut down the system.	Platform Instance Object
eShuttingDown	The system is shutting down.	Platform Instance Object
eSuspendRequested	A request has been sent out to suspend/hibernate the system.	Platform Instance Object
eSuspending	The system is suspending/hibernating.	Platform Instance Object
eCriticalResume	The system is resuming from suspension/hibernation. Before that, the system has been suspended/hibernated abnormally.	Platform Instance Object
eNormalResume	The system is resuming after being suspended/hibernated normally.	Platform Instance Object
eInternalPower	The power of the system has switched to DC.	Power Instance Object
eExternalPower	The power of the system has switched to AC.	Power Instance Object
eFullyCharged	The status of a battery or power provider has changed from **Charging** to fully **Charged**. To determine which type of instance object triggered the event, invoke the **GetObjectType()** method.	Power Instance Object / Battery Instance Object
eGoingOffline	The status of a battery has changed from powering the system to going offline.	Battery Instance Object
ePoweringSystem	The status of a battery has changed from going offline to powering the system.	Battery Instance Object
eBandwidthChanged	The bandwidth has changed.	Bandwidth Instance Object
eUnknown	An unknown event has occurred.	none

Methods

Return Value	Signature	Description
EventType	**GetType()**	Returns the type of event.
Datetime	**GetTimestamp()**	Returns the time the event was fired.
Void	**SetType(EventType type)**	Sets the type of the event. Called by the system.
Void	**SetTimestamp(Datetime timestamp)**	Sets the time the event fired. Called by the system.
String	**GetObjectType()**	Returns the type of the object that originated the event.
String	**GetObjectKey()**	Returns the key for the object that originated the event.
String	**GetParentKey()**	Returns the key for the parent of the object that originated the event, if any. Otherwise returns a NULL string.
String	**GetClassType()**	Returns a string containing the Class type of the object originating the event.
String	**GetTypeName()**	Returns a string containing the description of the event type.
String	**GetTSstring()**	Returns the time the event was fired as a string.
Object	**GetOriginator()**	Returns the object that originated the event if it is still in scope. Otherwise it returns NULL.

Remarks

Event objects are passed as an argument to the **Notify()** method of **Observer** derived objects. The class **Event** is only used in this context and **Event** objects are not persistent outside the scope of the **Observer Notify()** method.

Events can be originated by **ClassObjects**, **InstanceObjects**, **Properties** and **Thresholds**. Calling the **GetObjectType**() method will return a string describing the originating object type. The possible return values are:

- **ClassObject**
- **InstanceObject**
- **Property**
- **Threshold**

The method **GetObjectKey**() returns a value that is qualified by the object type:

- For **ClassObjects,** it returns the name of the class, such as “NetworkAdapter”.
- For **InstanceObjects,** it returns the instance key.
- For **Properties,** it returns the name of the Property.
- For **Thresholds,** it returns the name of the monitored Property.

The behavior of the **GetParentKey**() method is also dependent upon the type of the originating object:

- For **ClassObjects,** it returns NULL (an empty string)
- For **InstanceObjects,** it returns NULL (an empty string)
- For **Properties,** it returns the key for the **InstanceObject** that it is a member of.
- For **Thresholds,** it returns the key for the **InstanceObject** that the monitored Property is a member of.

The **GetClassType**() method returns the name of the class type the object belongs to, for example, “Battery”. In the case of Thresholds this method will return the class type of the property being monitored.

An application must register all relevant events for the items for which notifications are to be recieved. For example, to receive an event indicating that the power source has switched between internal and external power (see **Power**), the application must register for the **InternalPower** and **ExternalPower** events or the **Changed** event of the source. However, to receive an event indicating that the power source has switched to internal only, the application only needs to register for the **InternalPower** event.

See Also

Observer

CounterThreshold

A counter threshold is a class that sends a notification when the value of the counter (observed property) reaches or exceeds a comparison value.

C++

```
namespace Intel::Mobile::Threshold

class CounterThreshold : public Threshold
```

C#

```
namespace Intel.Mobile.Threshold

public class CounterThreshold : Threshold
```

Java

```
package com.intel.mobile.threshold

public abstract class CounterThreshold extends Threshold
```

Methods

Return Value	Signature	Description
Property	**GetObservedProperty()**	Get the property being observed.
void	**SetObservedProperty(** *[type]* **property)**	Set the observed property.
[type]	**GetThreshold()**	Get the threshold value
Boolean	**SetThreshold(** *[type]* **threshold)**	Set the threshold value.
[type]	**GetOffset()**	Get the increment value.
Boolean	**SetOffset(** *[type]* **offset)**	Set the increment value.
Boolean	**GetNotify()**	Get the notify flag. Returns **true** if the next crossing will fire an event.
Boolean	**SetNotify(bool notify)**	Set the notify flag. When set to **true**, the next crossing will fire an event.
int32	**GetGranularityPeriod()**	(msec) Get the granularity period.
Boolean	**SetGranularityPeriod(int32 nPeriod)**	(msec) Set the granularity period. ***Notes:*** • *Before setting the granularity period, call the* ***SetObservedProperty****.* • *The minimum value for granularity is 1000 msec (1 sec). If a lower value is entered, the property will be set to 1000.*
String	**GetType()**	Returns a string describing the threshold type. ***Note:*** *For the C++version, a returned string that is unused must be manually deleted to avoid a memory leak.*

Observer	**GetObserver(uint32 index)**	Returns the observer specified by the index value.
uint32	**GetObserverCount()**	Returns the total number of observers.
Boolean	**AddObserver(Observer** *observer* **)**	Adds an observer to the Threshold. When observed conditions match the target conditions, the **Notify()** method of the observed object is invoked. Returns **true** if successful.
Boolean	**RemoveObserver(Observer** *o* **)**	Removes an observer from the event. Returns **true** if successful.
Boolean	**RemoveObservers()**	Removes all observers (for the current application) from the event. Returns **true** if successful.
Boolean	**Start()**	Starts monitoring of the property value. Returns **true** if successful.
Boolean	**Stop()**	Stops monitoring of the property value. Returns **true** if successful.

Remarks

CounterThreshold is capable of observing integer type properties that behave like a counter, such as **byte**, **integer**, **unsigned integer, int64, unsigned int64,** and **DATE**. These properties have a value that is:

- Always greater than or equal to zero.
- Can only be incremented.

The offset property allows repeated alerts to be received. The offset defaults to zero, in which case a single **ThresholdReached** event will be received. At this time the framework will turn off notification. If further notifications are desired, one of two actions can be performed. Setting the notification by calling the **SetNotify()** method with an argument of true will result in a single additional notification to be received. At this time the framework will once again turn the notification off. If the **SetOffset()** method is called with a non-zero value, a notification will be received every time the observed value reaches “threshold+offset”. At that time, the framework increments the threshold value by the offset and will continue monitoring the value.

[type] represents the type of CounterThreshold, which can be **byte**, **integer**, **unsigned integer, int64, unsigned int64,** or **DATE**.

GaugeThreshold

A gauge threshold observes an attribute that is continuously variable with time. The gauge threshold provides buffering to avoid repeatedly triggering notifications when the value of a property varies within a range defined by high and low threshold values.

C++

```
namespace Intel::Mobile::Threshold

class GaugeThreshold : public Threshold
```

C#

```
namespace Intel.Mobile.Threshold

public class GaugeThreshold : Threshold
```

Java

```
package com.intel.mobile.threshold

public abstract class GaugeThreshold extends Threshold
```

Methods

Return Value	Signature	Description
Property	**GetObservedProperty()**	Get the property being observed.
void	**SetObservedProperty(** *[type]* **property)**	Set the observed property.
[type]	**GetHighThreshold()**	Get the high threshold value.
Boolean	**SetHighThreshold(** *[type]* **threshold)**	Set the high threshold value.
[type]	**GetLowThreshold()**	Get the low threshold value.
Boolean	**SetLowThreshold(** *[type]* **threshold)**	Set the low threshold value.
[type]	**GetOffset()**	Get the increment value.
Boolean	**SetOffset(** *[type]* **offset)**	Set the increment value.
Boolean	**GetNotifyHigh()**	Get the high notify flag. Returns **true** if the next high crossing will fire an event.
Boolean	**SetNotifyHigh(bool notifyHigh)**	Set the high notify flag. If set to **true**, the next high crossing will fire an event.
Boolean	**GetNotifyLow()**	Get the low notify flag. Returns **true** if the next low crossing will fire an event.
Boolean	**SetNotifyLow(bool notifyLow)**	Set the low notify flag. If set to **true**,the next low crossing will fire an event.
int32	**GetGranularityPeriod()**	(msec) Get the granularity period.
Boolean	**SetGranularityPeriod(int32 nPeriod)**	(msec) Set the granularity period. ***Note:*** • *Before setting the granularity period, call the* ***SetObservedProperty****.* • *The minimum value for granularity is 1000 msec (1 sec). If a lower value is entered, the property will be set to 1000.*
String	**GetType()**	Returns a string describing the threshold type. ***Note:*** *For the C++version, a returned string that is unused must be manually deleted to avoid a memory leak.*

Observer	**GetObserver(uint32 index)**	Returns the observer specified by the index value.
uint32	**GetObserverCount()**	Returns the total number of observers.
Boolean	**AddObserver(Observer** *observer* **)**	Adds an observer to the Threshold. When observed conditions match the target conditions, the **Notify()** method of the observed object is invoked. Returns **true** if successful.
Boolean	**RemoveObserver(Observer** *o* **)**	Removes an observer from the event. Returns **true** if successful.
Boolean	**RemoveObservers()**	Removes all observers (for the current application) from the event. Returns **true** if successful.
Boolean	**Start()**	Starts monitoring of the property value. Returns **true** if successful.
Boolean	**Stop()**	Stops monitoring of the property value. Returns **true** if successful.

Remarks

GaugeThreshold is capable of observing numeric type properties that can be monitored with a gauge, such as **byte**, **integer**, **unsigned integer, int64, unsigned int64, float** and **DATE**. These properties have a value that is constantly variable, either increasing or decreasing, or both.

Note: *It is the responsibility of the user to set the high and low threshold properties to appropriate values prior to calling the* ***Start*** *method. Typically, this means that the high value must be greater than the low value. If invalid values are used, the behavior is undefined.*

A gauge threshold sends notifications as follows:

- When the **NotifyHigh** property is set to **true**, and the observed property value increases and becomes equal to or greater than the high threshold value, a **ThresholdHigh** notification is sent.

  Subsequent crossings of the high threshold value do not cause further notifications unless the property value becomes equal to or less than the low threshold value.

- When the **NotifyLow** property is set to **true**, and the observed property value decreases and becomes equal to or less than the low threshold value, a **ThresholdLow** notification is sent.

  Subsequent crossings of the low threshold value do not cause further notifications unless the property value becomes equal to or greater than the high threshold value.

- If the **NotifyHigh** flag is set to **true**, the **NotifyLow** flag is automatically set to **true** when the observed property value reaches the high threshold. Setting the **NotifyLow** flag manually is not required and has no effect.

- If the **NotifyLow** flag is set to **true**, the **NotifyHigh** flag is automatically set to **true** when the observed property value reaches the low threshold. Setting the **NotifyHigh** flag manually is not required and has no effect.

The **Offset** property modifies the behavior of the system when a high or low threshold is reached:

- If the **Offset** is nonzero when the **HighThreshold** value is reached, the value of the **HighThreshold** is incremented by **Offset** and the **NotifyHigh** flag is left set to **true**.
- If the **Offset** is nonzero when the **LowThreshold** value is reached, the value of the **LowThreshold** is decremented by **Offset** and the **NotifyLow** flag is left set to **true**.

A derived gauge can be created to monitor the difference between the observed gauge values for two successive observations. The derived gauge value (`V[t]`) is calculated using the following method, where GP is the **GranularityPeriod**:

```
V[t] = gauge[t] - gauge[t-GP]
```

The threshold observer for a derived gauge requires the observed property to be one of the following types:

- byte
- integer
- short
- long
- float
- double

[type] represents the type of GaugeThreshold, which can be **byte**, **integer**, **unsigned integer, int64, unsigned int64,** or **DATE**.

The **GaugeThreshold** buffering mechanism used to prevent firing of multiple high or low threshold events presents a potential functionality issue.

For example, the level of charge in a battery could be monitored with the **HighThreshold** set to 90% and the **LowThreshold** 20%. If the user periodically attaches and detaches the system from its power supply, the charge level will vary non-linearly. If the user unplugs the system and allows the charge level to drop below 20%, the low threshold event will fire and the **NotifyLow** flag will be set to **false**. Consider what happens if the user then plugs in the system, leaves it attached until the charge level rises above 20%**,** and then unplugs it again before the charge level rises above the **High Threshold**. In this case, when the charge level drops below 20%, the **NotifyLow** event will not fire because the **NotifyLow** is set to **false**.

This issue can be resolved by using two Threshold objects, one to monitor the ‘discharged’ state and one to monitor the ‘fully charged’ state. The code samples below show how this could be accomplished.

```
class MyChargedObserver : public Observer
{
public:
    void Notify( const Event& event )
    {
        if ( event.GetType() == Event::eThresholdHigh )
        {
            printf( "The battery is charged" );
        }
    }
};
```

```
class MyDishargedObserver : public Observer
{
public:
    void Notify( const Event& event )
    {
        if ( event.GetType() == Event::eThresholdLow )
        {
            printf( "The battery is almost discharged" );
        }
    }
};
```

```
UInt64GaugeThreshold  myChargedThreshold;
UInt64GaugeThreshold  myDischargedThreshold;
```

```
MyChargedObserver     myChargedObserver;
MyDishargedObserver   myDischargedObserver;


myChargedThreshold.SetObservedProperty( pMyBattery-
       >LifePercentRemaining )
myChargedThreshold.SetHighThreshold   ( 90   );
myChargedThreshold.SetLowThreshold    ( 85   );
myChargedThreshold.SetGranularityPeriod( 5000 );
myChargedThreshold.AddObserver        ( myChargedObserver );


myDischargedThreshold.SetObservedProperty( pMyBattery-
       >LifePercentRemaining )
myDischargedThreshold.SetHighThreshold   ( 25   );
myDischargedThreshold.SetLowThreshold    ( 20   );
myDischargedThreshold.SetGranularityPeriod( 5000 );
myDischargedThreshold.AddObserver        ( myDischargedObserver );
```

ValueThreshold

ValueThreshold defines a threshold for the values of a string attribute.

C++

```
namespace Intel::Mobile::Threshold
class ValueThreshold : public Threshold
```

C#

```
namespace Intel.Mobile.Threshold
public class ValueThreshold : Threshold
```

Java

```
package com.intel.mobile.threshold
public abstract class ValueThreshold extends Threshold
```

Methods

Return Value	Signature	Description
Property	**GetObservedProperty()**	Get the property being observed.
void	**SetObservedProperty(** *[type]* **property)**	Set the observed property.
[type]	**GetValue()**	Get the threshold value.
Boolean	**SetValue(** *[type]* **threshold)**	Set the threshold value.
Boolean	**GetNotifyMatch()**	Get the notify high flag. Returns **true** if an event will be fired when the value changes to become equal to the threshold value.
Boolean	**SetNotifyMatch(bool notifyMatch)**	Set the notify flag. If set to **true**,an event will be fired when the value changes to become equal to the threshold value.
Boolean	**GetNotifyDiffer()**	Get the notify flag. Returns **true** if an event will be fired when the value changes to become different from the threshold value.
Boolean	**SetNotifyDiffer(bool notifyDiffer)**	Set the notify flag. If set to **true**,an event will be fired when the value changes to become different from the threshold value.
int32	**GetGranularityPeriod()**	(msec) Get the granularity period.
Boolean	**SetGranularityPeriod(int32 nPeriod)**	(msec)Set the granularity period. ***Note:*** • *Before setting the granularity period, call the* ***SetObservedProperty****.* • *The minimum value for granularity is 1000 msec (1 sec). If a lower value is entered, the property will be set to 1000.*
String	**GetType()**	Returns a string describing the threshold type. ***Note:*** *For C++ version, the returned string needs to be deleted manually when unused to avoid memory leak.*
Observer	**GetObserver(uint32 index)**	Returns the observer specified by the index value.
uint32	**GetObserverCount()**	Returns the total number of observers.
Boolean	**AddObserver(Observer** *o* **)**	Adds the observer to the event. When observed conditions match the target conditions, the **Notify()** method of the observed object is invoked.
Boolean	**RemoveObserver(Observer** *o* **)**	Removes the observer from the event.

Boolean	**RemoveObservers()**	Removes all observers (for the current application) from the event.
Boolean	**Start()**	Starts the threshold.
Boolean	**Stop()**	Stops the threshold.

Remarks

ValueThreshold is capable of observing string type properties that can have known values.

A value threshold sends notifications as follows:

- If the attribute value matches the comparison string, a **ThresholdEqual event** is fired. The **NotifyMatch** property must be set to **true**, using **SetNotifyMatch().**

  Subsequent matches of the string to the comparison value do not cause further events unless the attribute value differs from the comparison string.

- If the attribute value differs from the comparison string, a **ThresholdDiffer** event is fired. The **NotifyDiffer** property must be set to **true**.

  Subsequent differences from the string to the comparison string do not cause further notifications unless the attribute value matches the comparison string.

- If the **NotifyMatch** flag is set to **true**, the **NotifyDiffer** flag is automatically set to **true** when the observed property value matches the comparison threshold. Setting the **NotifyDiffer** flag manually is not required and has no effect.

- If the **NotifyDiffer** flag is set to **true**, the **NotifyMatch** flag is automatically set to **true** when the observed property value differs from the comparison threshold. Setting the **NotifyMatch** flag manually is not required and has no effect.

[type] represents the type of ValueThreshold. The only ValueThreshold type is String.

CounterThreshold

ByteCounterThreshold

The **CounterThreshold** class for use with properties of type **ByteProperty**.

C++

```
namespace Intel::Mobile::Threshold

class ByteCounterThreshold : public CounterThreshold
```

C#

```
namespace Intel.Mobile.Threshold

public class ByteCounterThreshold : CounterThreshold
```

Java

```
package com.intel.mobile.threshold

public class ByteCounterThreshold extends CounterThreshold
```

IntCounterThreshold

The **CounterThreshold** class for use with properties of type **IntProperty**.

C++

```
namespace Intel::Mobile::Threshold

class IntCounterThreshold : public CounterThreshold
```

C#

```
namespace Intel.Mobile.Threshold

public class IntCounterThreshold : CounterThreshold
```

Java

```
package com.intel.mobile.threshold

public class IntCounterThreshold extends CounterThreshold
```

UIntCounterThreshold

The **CounterThreshold** class for use with properties of type **UIntProperty**.

C++

```
namespace Intel::Mobile::Threshold

class UIntCounterThreshold : public CounterThreshold
```

C#

```
namespace Intel.Mobile.Threshold

public class UIntCounterThreshold : CounterThreshold
```

Java

```
package com.intel.mobile.threshold

public class UIntCounterThreshold extends CounterThreshold
```

Int64CounterThreshold

The **CounterThreshold** class for use with properties of type **Int64Property**.

C++

```
namespace Intel::Mobile::Threshold
class Int64CounterThreshold : public CounterThreshold
```

C#

```
namespace Intel.Mobile.Threshold
public class Int64CounterThreshold : CounterThreshold
```

Java

```
package com.intel.mobile.threshold
public class Int64CounterThreshold extends CounterThreshold
```

UInt64CounterThreshold

The **CounterThreshold** class for use with properties of type **UInt64Property**.

C++

```
namespace Intel::Mobile::Threshold
class uInt64CounterThreshold : public CounterThreshold
```

C#

```
namespace Intel.Mobile.Threshold
public class UInt64CounterThreshold : CounterThreshold
```

Java

```
package com.intel.mobile.threshold
public class UInt64CounterThreshold extends CounterThreshold
```

DateCounterThreshold

The **CounterThreshold** class for use with properties of type **DateProperty**.

C++

```
namespace Intel::Mobile::Threshold
class DateCounterThreshold : public CounterThreshold
```

C#

```
namespace Intel.Mobile.Threshold
public class DateCounterThreshold : CounterThreshold
```

Java

```
package com.intel.mobile. threshold

public class DateCounterThreshold extends CounterThreshold
```

GaugeThreshold

ByteGaugeThreshold

The **GaugeThreshold** class for use with properties of type **ByteProperty**.

C++

```
namespace Intel::Mobile::Threshold

class ByteGaugeThreshold : public GaugeThreshold
```

C#

```
namespace Intel.Mobile.Threshold

public class ByteGaugeThreshold : GaugeThreshold
```

Java

```
package com.intel.mobile.threshold

public class ByteGaugeThreshold extends GaugeThreshold
```

IntGaugeThreshold

The **GaugeThreshold** class for use with properties of type **IntProperty**.

C++

```
namespace Intel::Mobile::Threshold

class IntGaugeThreshold : public GaugeThreshold
```

C#

```
namespace Intel.Mobile.Threshold

public class IntGaugeThreshold : GaugeThreshold
```

Java

```
package com.intel.mobile.threshold

public class IntGaugeThreshold extends GaugeThreshold
```

UIntGaugeThreshold

The **GaugeThreshold** class for use with properties of type **UIntProperty**.

C++

```
namespace Intel::Mobile::Threshold

class UIntGaugeThreshold : public GaugeThreshold
```

C#

```
namespace Intel.Mobile.Threshold

public class UIntGaugeThreshold : GaugeThreshold
```

Java

```
package com.intel.mobile.threshold

public class UIntGaugeThreshold extends GaugeThreshold
```

Int64GaugeThreshold

The **GaugeThreshold** class for use with properties of type **Int64Property**.

C++

```
namespace Intel::Mobile::Threshold

class Int64GaugeThreshold : public GaugeThreshold
```

C#

```
namespace Intel.Mobile.Threshold

public class Int64GaugeThreshold : GaugeThreshold
```

Java

```
package com.intel.mobile.threshold

public class Int64GaugeThreshold extends GaugeThreshold
```

UInt64GaugeThreshold

The **GaugeThreshold** class for use with properties of type **UInt64Property**.

C++

```
namespace Intel::Mobile::Threshold

class UInt64GaugeThreshold : public GaugeThreshold
```

C#

```
namespace Intel.Mobile.Threshold

public class UInt64GaugeThreshold : GaugeThreshold
```

Java

```
package com.intel.mobile.threshold

public class UInt64GaugeThreshold extends GaugeThreshold
```

FloatGaugeThreshold

The **GaugeThreshold** class for use with properties of type **FloatProperty**.

C++

```
namespace Intel::Mobile::Threshold

class FloatGaugeThreshold : public GaugeThreshold
```

C#

```
namespace Intel.Mobile.Threshold

public class FloatGaugeThreshold : GaugeThreshold
```

Java

```
package com.intel.mobile.threshold

public class FloatGaugeThreshold extends GaugeThreshold
```

DateGaugeThreshold

The **GaugeThreshold** class for use with properties of type **DateProperty**.

C++

```
namespace Intel::Mobile::Threshold

class DateGaugeThreshold : public GaugeThreshold
```

C#

```
namespace Intel.Mobile.Threshold

public class DateGaugeThreshold : GaugeThreshold
```

Java

```
package com.intel.mobile.threshold

public class DateGaugeThreshold extends GaugeThreshold
```

ValueThreshold

StringValueThreshold

The **ValueThreshold** class for use with properties of type **StringProperty**.

C++

```
namespace Intel::Mobile::Threshold

class StringValueThreshold : public ValueThreshold
```

C#

```
namespace Intel.Mobile.Threshold

public class StringValueThreshold : ValueThreshold
```

Java

```
package com.intel.mobile.threshold

public class StringValueThreshold extends ValueThreshold
```

Class Library

This section describes the Intel® Mobile Platform SDK adaptation subsystem class library.

Methods, properties and events inherited from parent classes, or required by interface are included below.

Capabilities

Capability

This section describes the interfaces in the Capability namespace.

CapabilityClass

The Class object for capabilities.

C++

```
namespace Intel::Mobile::Capability

class CapabilityClass: ClassObject
```

C#

```
namespace Intel.Mobile.Capability

public class CapabilityClass: ClassObject
```

Java

```
package com.intel.mobile.capability

public class CapabilityClass extends ClassObject
```

CapabilityInstance

The Instance object for capabilities.

C++

```
namespace Intel::Mobile::Capability

class CapabilityInstance: public InstanceObject
```

C#

```
namespace Intel.Mobile.Capability

public class CapabilityInstance: InstanceObject
```

Java

```
package com.intel.mobile.capability

public class CapabilityInstance extends InstanceObject
```

CapabilityCollection

The Collection object for capabilities.

C++

```
namespace Intel::Mobile::Capability

class CapabilityCollection: public InstanceCollection
```

C#

```
namespace Intel.Mobile.Capability

public class CapabilityCollection: InstanceCollection
```

Java

```
package com.intel.mobile.capability

public class CapabilityCollection extends InstanceCollection
```

Connectivity

ConnectivityInstance

The instance object of type **ConnectivityInstance** is used to access system connectivity information. This and other capabilities are enumerated through the Capability class object. You can retrieve a capability, determine the type and access specific Connectivity functionality by treating the returned object as a **ConnectivityInstance**.

C++

```
namespace Intel::Mobile::Capability

class ConnectivityInstance: public CapabilityInstance
```

C#

```
namespace Intel.Mobile.Capability

public class ConnectivityInstance: CapabilityInstance
```

Java

```
package com.intel.mobile.capability

public class ConnectivityInstance extends CapabilityInstance
```

Properties

Name	Type	Get	Set	Static	Description
Connected	**Boolean**	X			**True** if the system has at least one valid network interface connected to a network. This does not necessarily mean that a given host is reachable. To determine if the host is reachable, the application should use **IsReachable()** instead.

Methods

Return Value	Signature	Description
Boolean	**IsReachable(String** *UriDestination* **)**	Returns **true** if the Uniform Resource Identifier (URI) is reachable. The URI is in the form of: **UriDestination = protocol** + "://" + **host** + [":"+port] + ["/"+**virtual_dir**] **protocol** = "tcp" | "ftp" | "http" | "icmp" Port *is required for "tcp"; It is optional for the others. Virtual directory only applies to "http".* **host = IPAddress** | **HostName** This method uses the platform's default proxy settings for HTTP and FTP.
String	**GetNetworkAdapter(String UriDestination)**	Returns the key of the **NetworkAdapter** that is currently used to communicate with the host. This string can be used to get an instance of a **NetworkAdapterInstance** object.
String	**GetNetworkLinkProtocol(String UriDestination)**	Returns the key of the **LinkProtocol** that is currently used to communicate the host. This string value can be used to get an instance of a **LinkProtocolInstance** object.
uint32	**GetLatency(String UriDestination)**	(msec) Get the latency to the host, where **UriDestination** is the URI of the destination. If the latency measurement times-out, because the destination could not be reached, the method returns the following: • For C++, **ConnectivityInstance::TIMEDOUT** • For CLR and Java, **ConnectivityInstance.TIMEDOUT** where **TIMEDOUT** = 0xFFFFFFFF

Events

Name	Description
Connect	Event is triggered to indicate that the system has gone from a disconnected state to having at least one connection. This event is not triggered if a new connection is made with a previous connection still intact.
Disconnect	Event is triggered to indicate that the system has lost its last connection and is now in a disconnected state. This event is not triggered if a system has multiple connections and just one of them is lost since it still has at least one connection available.
IpAddressChange	Event is triggered whenever a change occurs in the table that maps IP addresses to interfaces.
RouteTableChange	Event is triggered whenever the routing table changes.

Bandwidth

Bandwidth control is enforced at four levels in a system:

- System wide
- At the application level
- At the process level
- At the connection level

A Bandwidth Policy Manager shipped with the Intel® Mobile Platform SDK can be used to set bandwidth limits at any or all of these levels. An application can only control bandwidth limits for its own process and on any of its own connections.

BandwidthInstance

The Instance object for bandwidth.

C++

```
namespace Intel::Mobile::Capability

class BandwidthInstance: public CapabilityInstance
```

C#

```
namespace Intel.Mobile.Capability

public class BandwidthInstance: CapabilityInstance
```

Java

```
package com.intel.mobile.capability

public class BandwidthInstance extends CapabilityInstance
```

Properties

Name	Type	Get	Set	Static	Description
TheoreticalSystemRateTx	**float**	X			(Kbps) Returns the theoretical maximum send rate available to the system. When multiple links exist in a system at the same time, **TheoreticalSystemRateTx** is defined to be the fastest link. For example, if a system has a 100 Mbps Ethernet connection and a 11 Mbps 802.11b connection, 100 Mbps would be returned.
TheoreticalSystemRateRx	**float**	X			(Kbps) Returns the theoretical maximum receive rate available to the system. When multiple links exist in a system at the same time, the fastest link is returned.
FastestProtocol	**String**	X			(Kbps) Returns the connected protocol with the highest **TxRate** based on link speed. If a tie occurs, a wired protocol is chosen over a wireless protocol and a higher signal strength protocol is chosen over a lower signal strength protocol.
SystemRateTx	**float**	X			(Kbps) Returns the current aggregated **TxRate** based on the link speed of all connected protocols.
SystemRateRx	**float**	X			(Kbps) Returns the current aggregated **RxRate** based on the link speed of all connected protocols.
LimitTx	**float**	X	X		(Kbps) Set/get the maximum kilobits per second the calling process can transmit on network devices. This setting is overridden by application execution policy values.
LimitRx	**float**	X	X		(Kbps) Set/get the maximum kilobits per second the calling process can receive on network devices.

Name	Type	Get	Set	Static	Description
PercentTx	**float**	X	X		Set/get the maximum percentage of bandwidth that the calling process can transmit on network devices. Example: To set **PercentTx** to 90%, use: `PercentTx = 90;` (not `PercentTx = 0.9;`)
PercentRx	**float**	X	X		Set/get the maximum percentage of bandwidth that the calling process can receive over network devices.
RateTx	**float**	X			(Kbps) Get the measured transfer rate that the calling process is currently getting over network. This function returns a sampled average that is updated periodically.
RateRx	**float**	X			(Kbps) Get the measured receive rate that the calling process is currently getting over network. This function returns a sampled average that is updated periodically.
LimitedTx	**Boolean**	X	X		**true** indicates that the process currently has a send limit enforced. Otherwise, **false**.
LimitedRx	**Boolean**	X	X		**true** indicates that the process currently has a receive limit enforced. Otherwise, **false**.
Adaptive	**Boolean**	X	X		If **true**, the bandwidth limits automatically change proportionally to the change in the link speed. For example, if the link speed doubles, the limit doubles as well.
AdaptivelyLimitable	**Boolean**	X			If **true**, the limits for this process can be adaptive. If **false**, percent based functions shouldn't be used.

Methods

Return Value	Signature	Description
Boolean	**SetSessionLimitTx(SOCKET** *socketfd*, **float** *MaxKBitsPerSec*)	(Kbps) Sets the maximum kilobits per second the socket can transmit on network devices.
Boolean	**SetSessionLimitRx(SOCKET** *socketfd*, **float** *MaxKBitsPerSec*)	(Kbps) Sets the maximum kilobits per second the socket can receive on network devices.
Boolean	**SetSessionPercentTx(SOCKET** *socketfd*, **float** *MaxPercentage* **)**	(percent) Sets the maximum percentage of bandwidth the socket can transmit on network devices.
Boolean	**SetSessionPercentRx(SOCKET** *socketfd*, **float** *MaxPercentage* **)**	(percent) Sets the maximum percentage of bandwidth the socket can receive on network devices.
float	**GetSessionLimitTx(SOCKET** *socketfd* **)**	(Kbps) Gets the maximum kilobits per second the given socket can transmit over network devices.
float	**GetSessionLimitRx(SOCKET** *socketfd* **)**	(Kbps) Gets the maximum kilobits per second the socket can receive over network devices.
float	**GetSessionRateTx(SOCKET** *socketfd* **)**	(Kbps) Gets the measured transfer rate the connection has transmitted over network devices. This function returns a sampled average which is updated periodically.
float	**GetSessionRateRx(SOCKET** *socketfd* **)**	(Kbps) Gets the kilobits per second the socket has received over network devices. This function returns a sampled average, which is updated periodically.
float	**GetEffectiveSessionLimitTx(SOCKET** *socketfd* **)**	(bps) Gets the maximum bits per second the given socket can transmit over network devices. This function returns the “effective limit”, which is the lower of the limits set by the application execution policy, the process policy, or the system.
float	**GetEffectiveSessionLimitRx(SOCKET** *socketfd* **)**	(bps) Gets the maximum bits per second the socket can receive over network devices. This function returns the “effective limit”, which is the lower of the limits set by the application execution policy, the process policy, or the system.

Boolean	**GetSessionAdaptivelyLimitable(SOCKET** *socketid* **)**	If **true**, the limits for this connection can be adaptive. If **false**, percent-based functions shouldn't be used.
Boolean	**GetSessionAdaptive(SOCKET** *socketid* **)**	If **true**, the bandwidth limits automatically change proportional to the change in the link speed. For example, if the link speed doubles, the limit doubles as well.
Boolean	**SetSessionAdaptive(SOCKET** *socketid* **, boolean IsAdaptive)**	Turns the adaptive mode for a connection on or off. A setting **IsAdaptive = true**, turns on adaptive mode. It is turned off otherwise.
Boolean	**GetSessionLimitedRx(SOCKET** *socketid* **)**	If the return value is **true**, the limit for *socketid* is currently being enforced when receiving data.
Boolean	**GetSessionLimitedTx(SOCKET** *socketid* **)**	If the return value is **true**, the limit for *socketid* is currently being enforced when sending data.
Boolean	**SetSessionLimitedRx(SOCKET** *socketid* **, boolean IsLimited)**	Enables or disables enforcement of a connection limit for receiving data. Setting **IsLimited = true** turns on the limit enforcement. It is turned off otherwise.
Boolean	**SetSessionLimitedTx(SOCKET** *socketid* **, boolean IsLimited)**	Enables or disables enforcement of a connection limit for sending data. Setting **IsLimited = true** turns on the limit enforcement. It is turned off otherwise.

Events

Name	Description
BandwidthChanged	An event is triggered to indicate that the bandwidth of the system has changed. The event is triggered when the link speed of any connection on the system changes.

Power*

This section describes the object model for power.

PowerInstance

The interface to the power singleton (part of the capability Information/Event subsystem) is described below. This interface represents a system-wide perspective of power based on an aggregation of all power sources.

C++

```
namespace Intel::Mobile::Capability

class PowerInstance: public CapabilityInstance
```

C#

```
namespace Intel.Mobile.Capability

public class PowerInstance: CapabilityInstance
```

Java

```
package com.intel.mobile.capability

public class PowerInstance extends CapabilityInstance
```

PowerSourceType

The table below lists values for the **Source** property.

Value	Meaning
Internal	Indicates that the power is from an internal source (battery).
External	Indicates that the power is from an external source (AC/DC).

* For the Intel® Mobile Platform SDK 1.0, this item is not supported on the Windows Mobile Edition 2003 platform, but is expected to be supported in a future release.

Properties

Name	Type	Get	Set	Static	Description
Source	**PowerSourceType**	X			Internal or External.
Capacity	**uint64**	X			(mWh) Current capacity of the internal power sources.
InternalPowerStatus	**ConditionType**	X			**Charged** – All batteries are charged. **Charging** – At least one battery is charging. **Discharging** – At least one battery is discharging **Discharged** – All batteries completely discharged.
EstimatedTimeRemaining	**uint64**	X			(seconds) Remaining time of the DC power system taking into account statistical and historic data. This value may be identical to the **TimeRemainingAtCurrentRate** value, but is intended to provide a better estimate based on statistical and historic data.
FullInternalCapacity	**uint64**	X			(mWh) Current fully charged capacity of the internal power sources.
LifePercentRemaining	**uint32**	X			(%) Remaining percent of life of the DC power system. Ratio of current capacity to fully charged capacity in %. The value is 100 for a fully charged battery and 25 for a battery that has ¼ of the full capacity remaining.
InternalPowerRate	**int64**	X			(mW) Current rate of power change for the internal power sources. Discharge rate is a negative value. Charge rate is a positive value.
TimeRemainingAtCurrentRate	**uint64**	X			(seconds) Estimated time remaining when using internal power at the current rate of the discharge.

Events

Name	Description
InternalPower	Event is triggered when the system switches to an internal power source.
ExternalPower	Event is triggered when the system switches to an external power source.
FullyCharged	Event is triggered when all internal power sources are fully charged.

Devices

Battery

The Battery namespace provides access to information, policies, and events related to the system batteries. The information model for batteries is described below.

BatteryClass

The Class object for battery.

C++

```
namespace Intel::Mobile::Battery

class BatteryClass: public ClassObject
```

C#

```
namespace Intel.Mobile.Battery

public class BatteryClass: ClassObject
```

Java

```
package com.intel.mobile.battery

public class BatteryClass extends ClassObject
```

BatteryInstance

The Instance object for batteries.

C++

```
namespace Intel::Mobile::Battery

class BatteryInstance: public InstanceObject
```

C#

```
namespace Intel.Mobile.Battery

public class BatteryInstance: InstanceObject
```

Java

```
package com.intel.mobile.battery

public class BatteryInstance extends InstanceObject
```

ConditionType

The table below lists values for the **Condition** property.

Value	Meaning
Undetermined	Indicates that the battery is not being charged or discharged nor is the battery charged or discharged. On most cases, the battery is partly charged, i.e., the **LifePercentRemaining** is less than 100% but the **Rate** is zero.
Charging	Indicates that the battery is currently charging.
Discharging	Indicates that the battery is currently discharging.
Charged	Indicates that the battery is charged to maximum capacity.
Discharged	Indicates that the battery is discharged to minimum capacity and is no longer delivering power to the platform.

Properties

Name	Type	Get	Set	Static	Description
Capacity	**uint64**	X			(mWh) Current capacity of the battery.
Chemistry	**String**	X		X	An abbreviated character string that identifies the chemistry of the battery.
Condition	**Condition Type**	X			Current condition of the battery. See the **ConditionType** table above for a list of values.
CriticalAlert	**uint64**	X			(mWh) Capacity at which a **Critical** battery alert occurs. The value is set by the system power policy. The **Critical** event is triggered when the capacity drops to the **CriticalAlert** threshold.
CriticalBias*	**uint64**	X		X	(mWh) A bias relative to zero, which is applied to battery reporting results. Some batteries reserve a small charge that is biased out of battery capacity values to show "0" as the critical battery level.
CycleCount*	**uint64**	X			Cycle count of the battery. The number of charge/discharge cycles the battery has experienced. This provides an indication of battery wear.
DesignedCapacity*	**uint64**	X		X	(mWh) Theoretical capacity of the battery when new.

* Only applicable on Windows XP Professional.

Name	Type	Get	Set	Static	Description
EstimatedTimeRemaining	**uint64**	X			(seconds) Remaining time of the battery taking into account statistical and historic data. This value may be identical to the value of **TimeRemainingAtCurrentRate**, but is intended to provide a better estimate based on statistical and historic data.
FullChargedCapacity	**uint64**	X			(mWh) Current fully charged capacity of the battery.
ID	**String**	X		X	Unique ID of the battery.
InternalStatusUpdateInterval	**uint64**	X			(mseconds) Interval at which the battery internally updates its status. ***Note:*** *The value of the* ***Condition*** *property must be obtained first before getting this value or the value of this property will be NULL.*
LifePercentRemaining	**uint32**	X			(%) Remaining percent of battery life. Ratio of current capacity to fully charged capacity in %. The value is 100 for a fully charged battery and 25 for a battery that has ¼ of full capacity remaining.
LowAlert	**uint64**	X			(mWh) Capacity, at which a **Low** battery alert occurs. The value is set by the system power policy. The **Low** event is triggered when capacity drops to the **LowAlert** threshold.
ManufactureDate*	**DateTime**	X		X	Manufacture date of the battery.
Manufacturer*	**String**	X		X	Name of the battery manufacturer.
Name*	**String**	X		X	Name of the battery.
Rate	**int64**	X			(mW) Current rate of the battery. Discharge rate is a negative value. Charge rate is a positive value.
Rechargeable	**Boolean**	X		X	**true** if the battery technology is rechargeable. **false** if the battery technology is non-rechargeable.

* Only applicable on Windows XP Professional.

Name	Type	Get	Set	Static	Description
ReportingScaleCapacity*	**uint64 [4]**	X			(mWh) Upper capacity limit for granularity. The value of granularity is valid for capacities reported by the battery that are less than or equal to this capacity but greater than or equal to the capacity given in the previous array element (or zero if this is the first array element). This property is used together with **ReportingScaleGranularity** and **ReportingScaleEntries**.
ReportingScaleEntries*	**uint32**	X			Number of entries in the reporting scale. The maximum is 4. This property is used together with **ReportingScaleCapacity** and **ReportingScaleGranularity**.
ReportingScaleGranularity*	**uint64 [4]**	X			(mWh) Granularity of the capacity reading returned by the battery. Granularity may change over time as battery discharge and recharge lowers the range of readings. This property is used together with **ReportingScaleCapacity** and **ReportingScaleEntries**.
SerialNumber*	**String**	X		X	Serial number of the battery.

* Only applicable on Windows XP Professional.

Name	Type	Get	Set	Static	Description
SuggestedCriticalAlert*	**uint64**	X		X	(mWh) Manufacturer's suggestion of a capacity at which an ACPI 2.0-conforming "low" battery alert should occur (I.e., DefaultAlert1 as defined in the *Microsoft* Platform Software Development Kit*) Definitions of "low" battery vary from manufacturer to manufacturer. A "warning" state will typically occur before a "low" state, but may not always. To reduce risk of data loss, this value is usually used as the default setting for the critical battery alarm. When capacity drops to the **SuggestedCriticalAlert** threshold, the **SuggestedCritical** event is triggered.
SuggestedLowAlert*	**uint64**	X		X	(mWh) Manufacturer's suggestion of a capacity at which an ACPI 2.0-conforming "warning" battery alert should occur (i.e., DefaultAlert2 as defined in the *Microsoft* Platform Software Development Kit*) Definitions of "warning" vary from manufacturer to manufacturer. A "warning" state will typically occur before a "low" state, but may not always. To reduce risk of data loss, this value is usually used as the default setting for the low battery alarm. When capacity drops to the **SuggestedLowAlert** threshold, the **SuggestedLow** event is triggered.
SystemBattery	**Boolean**	X		X	**true** indicates that the battery can provide general power to run the system. **false** indicates that the normal operation is for a fail-safe function only (i.e., battery is not expected to be used during normal system usage).

* Only applicable on Windows XP Professional.

Name	Type	Get	Set	Static	Description
Temperature	**uint32**	X			(10th of K) temperature of the battery. Temperature (T) in °C = T/10 - 273.15 Temperature (T) in °F = (T/10 - 273.15) * 9/5 + 32 For example, 3237 = 323.7 K = 50.5 C = 123 F
TimeRemainingAtCurrentRate	**uint64**	X			(seconds) Estimated time remaining at the current **Rate** of the battery. This value is not very accurate on some battery systems.
Voltage	**uint64**	X			(mV) Current voltage of the battery.

Events

Name	Description
FullyCharged	Event is triggered when a battery has completed charging and is now fully charged.
GoingOffline	Event is triggered when a battery is going offline and is no longer powering the system. Either another battery is now powering the system or the system is on AC power. At any given time, only one battery is powering the system, thus only one event is triggered when the system goes on AC power.
PoweringSystem	Event is triggered when a battery is starting to power the system.

Remarks

When a battery is removed from the system, the device itself (the battery bay) is still considered to be present in the system.

Strictly speaking, mA and mAh are units for rate and capacity, while mW and mWh are units for power and energy respectively. However, the Battery namespace uses the more commonly-used terms rate and capacity with the understanding that the physical properties are power and energy as denoted by the units of the **Rate** (mW) and **Capacity** (mWh) properties.

The *Advanced Configuration And Power Interface Specification Revision 2.0c* (ACPI 2.0) specifies “warning”, “low” and “critical” levels for the battery. To be on the conservative side and to reduce the risk of data loss, the definitions used in the Battery namespace differ from the ACPI 2.0 specification and correspond to the specification as follows:

- The ACPI 2.0 “warning” level corresponds to a “low” threshold in the Battery namespace. For example, this value is used for the **LowAlert** property and the **Low** event.

- The ACPI 2.0 “low” level corresponds to a “critical” threshold in the Battery namespace. For example, this value is used for the **LowAlert** property and the **Low** event.

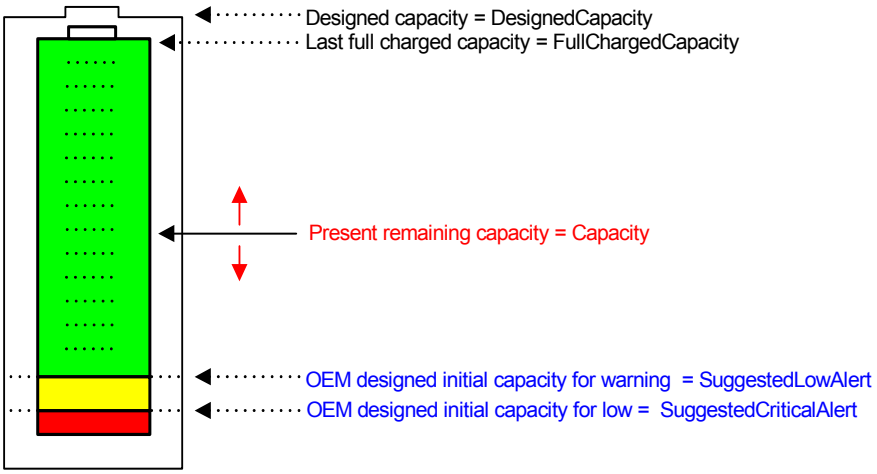

Figure 5: Reporting battery capacity – correlation to ACPI 2.0 standard

A "low” state typically occurs before a "critical" state, but not always. It is possible to poll a battery to find that neither alert level has occurred and then poll the battery again to find it discharged to the extent that both levels have been achieved. This may indicate that you are not polling often enough. It may also indicate that the battery is unable to hold a charge for very long and is discharging more rapidly than you expected. Such a battery may be nearing the end of its useful life, or it may be damaged.

BatteryCollection

The Collection object for batteries.

C++

```
namespace Intel::Mobile::Battery

class BatteryCollection: public InstanceCollection
```

C#

```
namespace Intel.Mobile.Battery

public class BatteryCollection: InstanceCollection
```

Java

```
package com.intel.mobile.battery

public class BatteryCollection extends InstanceCollection
```

Network

This section describes the interfaces in the Network namespace.

Model Organization

The Network model is composed of two main parts: the Link Protocol and the Network Adapter. Their relationships are shown in Figure 6.

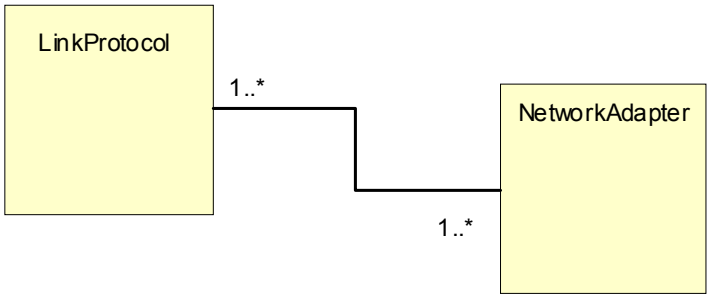

Figure 6. Network schema

The Link Protocol (**LinkProtocol**) describes the link protocols running on the system, while the Network Adapter (**NetworkAdapter**) describes the network adapters currently available in the system.

***Note**: Network Adapters and Link Protocols have a many-to-many relationship. In other words, a given link protocol, such as 802.11a, may be provided by more than one network adapter and a given network adapter may provide more that one link protocol, such as 802.11a and GPRS.*

NetworkAdapter

The Adapter schema is used to describe network adapter devices present in the system. Properties in this schema relate to the physical aspects of devices rather than the protocols active on the devices.

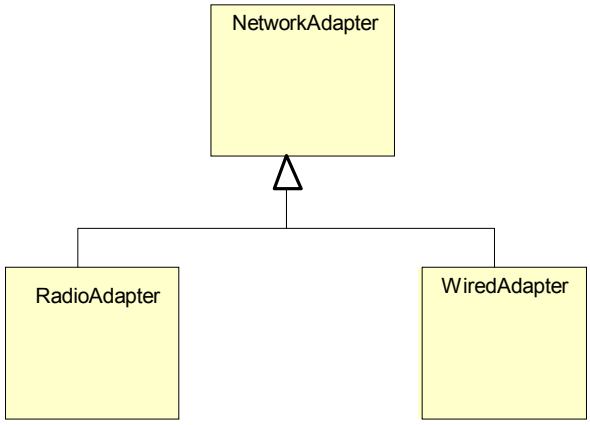

Figure 7. Adapters

NetworkAdapterClass

The Class object for network adapters.

C++

```
namespace Intel::Mobile::Network

class NetworkAdapterClass: ClassObject
```

C#

```
namespace Intel.Mobile.Network

public class NetworkAdapterClass: ClassObject
```

Java

```
package com.intel.mobile.network

public class NetworkAdapterClass extends ClassObject
```

NetworkAdapterInstance

The Instance object for network adapters. The **NetworkAdapter** interface provides the basic structure for organizing network adapter-related information in the system.

C++

```
namespace Intel::Mobile::Network

class NetworkAdapterInstance: public InstanceObject
```

C#

```
namespace Intel.Mobile.Network

public class NetworkAdapterInstance: InstanceObject
```

Java

```
package com.intel.mobile.network

public class NetworkAdapterInstance extends InstanceObject
```

NetworkAdapterType

The table below lists values for the **AdapterType** property.

Value	Meaning
0	Unspecified
1	LinkProtocol802_3
2	GPRS
3 *	CDMA
4	LinkProtocol802_11
5	IrDa
6	IEEE1394
7	PhoneLine
8 *	Bluetooth
9 *	UWB

NetConnectionStatusType

The table below lists values for the **NetConnectionStatusType** property.

Value	Meaning
Disconnected	The network adapter is disconnected.
Connecting	The network adapter is currently in the process of connecting to a network.
Connected	The network adapter is currently connected.
Disconnecting	The network adapter is currently disconnecting.
HardwareNotPresent	The network adapter is not currently present in the system.
HardwareDisabled	The network adapter is present but currently disabled.
HardwareMalfunction	The network adapter is malfunctioning.
Authenticating	The network adapter is connected but not yet usable as it is part way through the authentication process.
AuthenticationSucceeded	The network adapter is connected and the authentication process is completed.
AuthenticationFailed	The network adapter is connected but not usable since the authentication process failed.
Unspecified	The connection status was not available.

* *Not supported in the Intel® Mobile Platform SDK 1.0. Reserved for future use.*

* *Not supported in the Intel® Mobile Platform SDK 1.0. Reserved for future use.*

Properties

Name	Type	Get	Set	Static	Description
AdapterType	**NetworkAdapterType**	X			Numeric value describing what type of adapter the device is. The values are described in the **NetworkAdapterType** table above.
AutoSense	**Boolean**	X			If **true**, the network adapter is capable of automatically determining the speed of the attached/network media.
Description	**String**	X			Description of the object, such as “Intel® PRO/Wireless LAN 2100 3B Mini PCI Adapter”.
DeviceId	**String**	X			Unique identifier of the network adapter on the system.
Enabled	**Boolean**	X			Users or systems can enable and disable hardware through software, such as Plug and Play disabling software. This value reflects whether the device is enabled or not.
Index	**int32**	X			Unique non-sequential numeric index of the device as reported by the operating system.
Manufacturer	**String**	X		X	Name of the manufacturer of the network adapter, such as “Intel”.
Name	**String**	X			The name of the device, such as “Intel® PRO/Wireless LAN 2100 3B Mini PCI Adapter”. Used for device identification.
NdiLower*	**String**	X			The lower NDI object bound to the device, such as “Ethernet”.
NdiUpper*	**String**	X			The upper NDI object bound to the device, such as “Ndis5, wifipro”.
NetConnectionId*	**String**	X			Name of the network connection as it appears in the Windows Network Connections tool (for example, “Local Area Connection”). In Windows XP, this tool is accessible at **Start > Control Panel > NetworkConnections**.

* Only applicable on Windows XP Professional.

NetConnectionStatus	**NetConnectionStatus Type**	X	State of the network adapter connection to the network.
PnpDeviceId	**String**	X	Plug and Play device identifier of the logical device.
Protocols	**StringArray**	X	Array of keys that apply to the Intel® Mobile Platform SDK protocols that this device manages.

NetworkAdapterCollection

The Collection object for network adapters.

C++

```
namespace Intel::Mobile::Network

class NetworkAdapterCollection: public InstanceCollection
```

C#

```
namespace Intel.Mobile.Network

public class NetworkAdapterCollection: InstanceCollection
```

Java

```
package com.intel.mobile.network

public class NetworkAdapterCollection extends InstanceCollection
```

WiredAdapterClass

The Class object for wired network adapters.

C++

```
namespace Intel::Mobile::Network

class WiredAdapterClass: ClassObject
```

C#

```
namespace Intel.Mobile.Network

public class WiredAdapterClass: ClassObject
```

Java

```
package com.intel.mobile.network

public class WiredAdapterClass extends ClassObject
```

WiredAdapterInstance

The **WiredAdapter** provides the basic structure for organizing network adapters that do not use a radio.

Note: *Because* **WiredAdapter** *is inherited from* **NetworkAdapter***, all the methods, properties, and events in* **NetworkAdapter** *are available in* **WiredAdapter***.*

C++

```
namespace Intel::Mobile::Network

class WiredAdapterInstance: public NetworkAdapterInstance
```

C#

```
namespace Intel.Mobile.Network

public class WiredAdapterInstance: NetworkAdapterInstance
```

Java

```
package com.intel.mobile.network

public class WiredAdapterInstance extends NetworkAdapterInstance
```

WiredAdapterCollection

The Collection object for wired network adapters.

C++

```
namespace Intel::Mobile::Network

class WiredAdapterCollection: public InstanceCollection
```

C#

```
namespace Intel.Mobile.Network

public class WiredAdapterCollection: InstanceCollection
```

Java

```
package com.intel.mobile.network

public class WiredAdapterCollection extends InstanceCollection
```

RadioAdapterClass

The Class object for radio-based network adapters.

C++

```
namespace Intel::Mobile::Network

class RadioAdapterClass: ClassObject
```

C#

```
namespace Intel.Mobile.Network

public class RadioAdapterClass: ClassObject
```

Java

```
package com.intel.mobile.network

public class RadioAdapterClass extends ClassObject
```

RadioAdapterInstance

The Instance object for radio-based network adapters.

C++

```
namespace Intel::Mobile::Network

class RadioAdapterInstance: public NetworkAdapterInstance
```

C#

```
namespace Intel.Mobile.Network

public class RadioAdapterInstance: NetworkAdapterInstance
```

Java

```
package com.intel.mobile.network

public class RadioAdapterInstance extends NetworkAdapterInstance
```

RadioAdapterCollection

The Collection object for radio-based network adapters.

C++

```
namespace Intel::Mobile::Network

class RadioAdapterCollection: public InstanceCollection
```

C#

```
namespace Intel.Mobile.Network

public class RadioAdapterCollection: InstanceCollection
```

Java

```
package com.intel.mobile.network

public class RadioAdapterCollection extends InstanceCollection
```

LinkProtocol

The Protocol schema specifies OSI Layer 2 and Layer 3 properties of protocols.

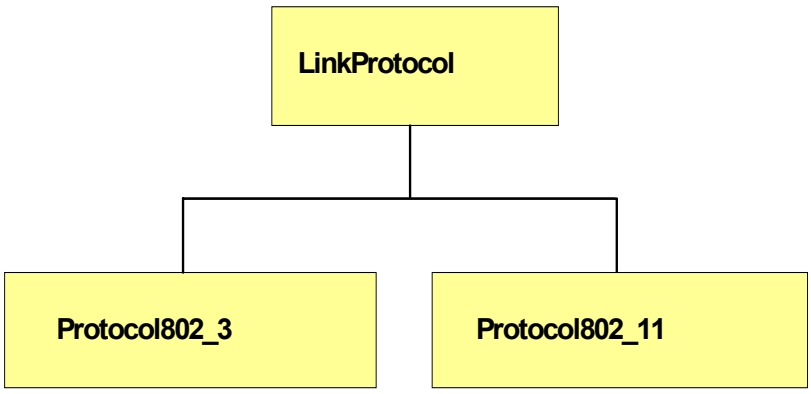

Figure 8. Protocols

LinkProtocolClass

The Class object for link protocols.

C++

```
namespace Intel::Mobile::Network

class LinkProtocolClass: ClassObject
```

C#

```
namespace Intel.Mobile.Network

public class LinkProtocolClass: ClassObject
```

Java

```
package com.intel.mobile.network

public class LinkProtocolClass extends ClassObject
```

LinkProtocolInstance

The Instance object for link protocols.

C++

```
namespace Intel::Mobile::Network

class LinkProtocolInstance: public InstanceObject
```

C#

```
namespace Intel.Mobile.Network

public class LinkProtocolInstance: InstanceObject
```

Java

```
package com.intel.mobile.network

public class LinkProtocolInstance extends InstanceObject
```

ProtocolStatusType

Value	Description
DeviceNotPresent	Device not present (either disabled or not physically present).
DevicePresent	Device present.
MediaPresent	Media present.
NonZeroIPAddress	Non-zero IP address.

Properties

Name	Type	Get	Set	Static	Description
CurrentRxDataRate	**uint64**				(bps) The current receive data rate.
CurrentTxDataRate	**uint64**				(bps) The current transmit data rate.
Enabled	**Boolean**	X			Indicates whether or not the protocol is available to applications. This is determined primarily by the status of the hardware to which the particular protocol applies.
Id	**String**	X		X	Uniquely identifies the protocol instance based on the protocol type and the **DeviceId** property of the device.
IpAddress	**String**	X			The IP address. Both IPv4 and IPv6 are supported.
MaxLinkSpeed	**uint64**	X		X	(bps) The maximum speed of the link.
MediaPresent	**Boolean**	X			**true** if the protocol is connected to a type of media.
Status	**ProtocolStatusType**	X			The state of the network adapter's connection to the network.

SupportedOIDs	**uint32Array**	X	X	Array of protocol-supported NDIS OIDs. Useful for getting additional access to the device.
SupportedOIDsDescriptions	**StringArray**	X	X	Array of protocol-supported NDIS OID descriptions. Useful for displaying in readable format.
SupportedRxDataRates	**uint64Array**	X	X	(bps) Array of supported receive rates.
SupportedTxDataRates	**uint64Array**	X	X	(bps) Array of supported transit rates.
TxPower	**uint32**	X		(mW) Transmit power.

Methods

Return Value	Signature	Description
String	**GetDevice ()**	Returns the key of the network adapter device that is associated with this protocol instance.

Events

Signature	Description
MediaConnect	Event is triggered when any protocol senses attached media and all protocols previously had no attached media.
MediaDisconnect	Event is triggered when the IProtocol becomes disconnected to media when a cable is removed.

LinkProtocolCollection

The Collection object for link protocols.

C++

```
namespace Intel::Mobile::Network

class LinkProtocolCollection: public InstanceCollection
```

C#

```
namespace Intel.Mobile.Network

public class LinkProtocolCollection: InstanceCollection
```

Java

```
package com.intel.mobile.network

public class LinkProtocolCollection extends InstanceCollection
```

Protocol802_3Class

The Class object for 802.3 protocols.

C++

```
namespace Intel::Mobile::Network

class Protocol802_3Class: ClassObject
```

C#

```
namespace Intel.Mobile.Network

public class Protocol802_3Class: ClassObject
```

Java

```
package com.intel.mobile.network

public class Protocol802_3Class extends ClassObject
```

Protocol802_3Instance

The Instance object for 802.3 protocols.

C++

```
namespace Intel::Mobile::Network

class Protocol802_3Instance: public LinkProtocolInstance
```

C#

```
namespace Intel.Mobile.Network

public class Protocol802_3Instance: LinkProtocolInstance
```

Java

```
package com.intel.mobile.network

public class Protocol802_3Instance extends LinkProtocolInstance
```

Properties

Name	Type	Get	Set	Static	Description
MacAddress*	**byte[6]**	X		X	Media access control address for this network adapter. A MAC address is a unique 48-bit number assigned to the network adapter by the manufacturer. It uniquely identifies the network adapter and is used for mapping TCP/IP network communications. If the device is not present, this value is empty.
MulticastAddress*	**StringArray**	X			Array of IP addresses that are bound to the protocol as multicast addresses.

* For the Intel® Mobile Platform SDK 1.0, this item is not supported on the Windows Mobile Edition 2003 platform, but is expected to be supported in a future release.

Protocol802_3Collection

The Collection object for 802.3 protocols.

C++

```
namespace Intel::Mobile::Network

class Protocol802_3Collection: public InstanceCollection
```

C#

```
namespace Intel.Mobile Network

public class Protocol802_3Collection: InstanceCollection
```

Java

```
package com.intel.mobile.network

public class Protocol802_3Collection extends InstanceCollection
```

Protocol802_11Class

The Class object for 802.11 protocols.

C++

```
namespace Intel::Mobile::Network

class Protocol802_11Class: ClassObject
```

C#

```
namespace Intel.Mobile.Network

public class Protocol802_11Class: ClassObject
```

Java

```
package com.intel.mobile.network

public class Protocol802_11Class extends ClassObject
```

Protocol802_11Instance

The Instance object for 802.11 protocols.

C++

```
namespace Intel::Mobile::Network

class Protocol802_11Instance: public LinkProtocolInstance
```

C#

```
namespace Intel.Mobile.Network

public class Protocol802_11Instance: LinkProtocolInstance
```

Java

```
package com.intel.mobile.network

public class Protocol802_11Instance extends LinkProtocolInstance
```

ProtocolModeType

Note: *This type only available for LinkProtocl802_11.*

Value	Description
AdHoc	Specifies the independent basic service set (IBSS) mode. This mode is also known as ad hoc mode.
Infrastructure	Specifies the infrastructure mode.
Auto	Specifies an automatic mode. In this mode, the 802.11 NIC can switch between ad hoc and infrastructure modes as required.

ProtocolPowerType

Value	Description
ContinuousAccess	Specifies continuous access mode (CAM). When the power mode is set to CAM, the device is always on
FastPowerSaving	This power mode provides the best combination of network performance and power usage
MaxPowerSaving	Specifies maximum (MAX) power savings. A power mode of MAX results in the greatest power savings for the 802.11 NIC radio.

Properties

Name	Type	Get	Set	Static	Description
ATIMWindow	**uint32**	X			(Kμsec, 1024 μsec) The announcement traffic information message (ATIM) window. This property is valid only in the Ad Hoc mode.
BeaconPeriod	**uint32**	X			(Kμsec, 1024 μsec) The interval between beacon message transmissions.
BssId	**byte[6]**	X			The MAC address of the access point with which the station is associated.
CurrentFrequency	**uint32**	X			(kHz) Specifies the frequency of the current channel.
DwellTime	**uint32**	X			(TU) The preset time period during transmission while operating in a single channel.

Name	Type	Get	Set	Static	Description
FailedCount	**uint64**	X			The number of frame transmissions that have failed after exceeding either the short frame or the long frame retry limits.
FCSErrorCount	**uint64**	X			The number of received frames with FCS errors.
FragmentationThreshold	**uint32**	X			(bytes) The fragmentation threshold.
FrameDuplicateCount	**uint64**	X			The number of duplicate frames received.
HopPattern	**uint32**	X			The current pattern being used for frequency hopping.
HopSet	**uint32**	X			The current set of hopping patterns being used.
InfrastructureModeId	**ProtocolModeType**	X			The current infrastructure mode that the station is operating in.
MacAddress	**byte[6]**	X			The MAC address of the station.
MulticastAddress	**StringArray**	X			Array of IP addresses that are bound to the protocol as multicast addresses.
MulticastReceivedFrameCount	**uint64**	X			The number of multicast or broadcast frames received.
MulticastTransmittedFrameCount	**uint64**	X			The number of frames that have been transmitted by multicast or broadcast.
MultipleRetryCount	**uint64**	X			The number of frames successfully retransmitted after more than one retransmission attempt.

Name	Type	Get	Set	Static	Description
NetworkTypeInUse	**int32**	X			The current physical layer network subtype that the 802.11 NIC and driver are using: **0** Ndis802_11FH, Frequency-hopping spread-spectrum radio **1** Ndis802_11DS, Direct sequencing spread-spectrum radio (802.11b,g) **2** Ndis802_11OFDM5, 5GHz OFDM radios (802.11g) **3** Ndis802_11OFDM24, 2.4GHz OFDM radios (802.11a/b/g)
NetworkTypesSupported	**int32Array**	X		X	An array of all the physical layer network subtypes that the 802.11 NIC and the driver support: **0** Ndis802_11FH, Frequency-hopping spread-spectrum radio **1** Ndis802_11DS, Direct sequencing spread-spectrum radio (802.11b,g) **2** Ndis802_11OFDM5, 5GHz OFDM radios (802.11g) **3** Ndis802_11OFDM24, 2.4GHz OFDM radios (802.11a/b/g)

Name	Type	Get	Set	Static	Description
PowerMode	**ProtocolPowerType**	X	X		Defines the current power mode. Values are: **1** Continuous Access Mode **2** Max Power Saving Mode **3** Fast Power Saving Mode ***Note:*** *Setting the* ***PowerMode*** *value successfully is dependent on the driver. After invoking* **SetValue()***, check the return value to verify success.*
ReceivedFragmentCount	**uint64**	X			The number of fragments received successfully.
RetryCount	**uint64**	X			The number of frames successfully retransmitted after one or more retransmission attempts.
Rssi	**int32**	X			(dBm) The received signal strength indication (RSSI).
RTSFailureCount	**uint64**	X			The number of times a CTS has not been received in response to an RTS.
RTSSuccessCount	**uint64**	X			The number of times a CTS has been received in response to an RTS.
RxAntenna	**uint32**	X	X		A number that identifies the current receive antenna. ***Note:*** *Setting the* **RxAntenna** *value successfully is dependent on the driver. After invoking* **SetValue()***, check the return value to verify success.*
SSID	**String**	X	X		The station's service set identifier (SSID).

Name	Type	Get	Set	Static	Description
TransmittedFragmentCount	**uint64**	X			The number of data and management fragments that have been successfully transmitted.
TxAntenna	**uint32**	X	X		A number that identifies the current transmit antenna. ***Note:*** *Setting the* **TxAntenna** *value successfully is dependent on the driver. After invoking* **SetValue()***, check the return value to verify success.*

Methods

Return Value	Signature	Description
Boolean	**Is80211a()**	Determines if IProtocol is type 802.11a protocol.
Boolean	**Is80211b()**	Determines if IProtocol is type 802.11b protocol.
Boolean	**Is80211g()**	Determines if IProtocol is type 802.11g protocol.

Protocol802_11Collection

The Collection object for 802.11 protocols.

C++

```
namespace Intel::Mobile::Network

class Protocol802_11Collection: public InstanceCollection
```

C#

```
namespace Intel.Mobile.Network

public class Protocol802_11Collection: InstanceCollection
```

Java

```
package com.intel.mobile.network

public class Protocol802_11Collection extends InstanceCollection
```

Platform*

The Platform namespace deals with hardware-based information required by the Adaptation subsystem that is not provided elsewhere, such as sleep, standby, and restore transitions and lid open and closed events.

PlatformClass

The Class object for platform.

C++

```
namespace Intel::Mobile::Platform

class PlatformClass: public ClassObject
```

C#

```
namespace Intel.Mobile.Platform

public class PlatformClass: ClassObject
```

Java

```
package com.intel.mobile.platform

public class PlatformClass extends ClassObject
```

PlatformInstance

The Instance object for platform.

C++

```
namespace Intel::Mobile::Platform

class PlatformInstance: public InstanceObject
```

C#

```
namespace Intel.Mobile.Platform

public class PlatformInstance: InstanceObject
```

Java

```
package com.intel.mobile.platform

public class PlatformInstance extends InstanceObject
```

* For the Intel® Mobile Platform SDK 1.0 release, this item is not supported on the Windows Mobile Edition 2003 platform, but is expected to be supported in a future release.

Methods

Return Value	Signature	Description
Boolean	**RequestShutDown()**	Allows the caller to request a system shutdown. Returns **true** asynchronously if the request is allowed and the system is now shutting down. The calling process must be prepared for shut down before making this call because the call may not return before the process is forcibly terminated. Returns **false** if the request is denied for any reason.
Boolean	**RequestSuspend()**	Allows the caller to request a system suspend. If the request is allowed, the system will immediately suspend and the call will not return until the system resumes, at which point, the return value will be **true**. Calling process must be prepared for suspension before making this call because the call *will not* return before the system suspends. Returns **false** if the request is denied for any reason.
Boolean	**RequestHibernate()**	Allows the caller to request system hibernation. If the request is allowed, the system will immediately hibernate and the call will not return until the system resumes, at which point, the return value will be **true**. The calling process must be prepared for hibernation before making this call because the call *will not* return before the system suspends. Returns **false** if the request is denied for any reason.

Events

Name	Description
ShutDownRequested	Event is triggered when any process on the system requests a system shutdown operation. If **RequestShutDown** is performed successfully, a subsequent **ShuttingDown** event is triggered.
SuspendRequested	Event is triggered when any process on the system requests a system suspend operation. If **RequestSuspend** is performed successfully, a subsequent **Suspending** event is triggered.
ShuttingDown	Event is triggered when the system begins to shut down. After this event has been triggered, subscribing applications should consider powering down to be inevitable.
Suspending	Event is triggered when the system begins suspending. After this event has been triggered, subscribing applications should consider suspending to be inevitable.
CriticalResume	Event is triggered when the system was hibernated or suspended without warning. If this event is received, the system was forced to suspend or resume without triggering the **Suspend/HibernateRequested** or **Suspending/Hibernating** events to allow an application to save its state. For user session independent processes, such as system services, this event is also triggered when the system is resuming from a restart or shut down before which no **ShutDown/RestartRequested** or **ShuttingDown/Restarting** events were triggered to allow the process to save its state.

Name	Description
NormalResume	Event is triggered when the system is resuming from hibernation or suspension in which **Suspend/HibernateRequested** and **Suspending/Hibernating** events were triggered. For user session independent processes, such as system services, this event will also be triggered when the system is resuming from a restart or shutdown before which **ShutDown/RestartRequested** or **ShuttingDown/Restarting** events were triggered.

PlatformCollection

The Collection object for platform.

C++

```
namespace Intel::Mobile::Platform

class PlatformCollection: public InstanceCollection
```

C#

```
namespace Intel.Mobile.Platform

public class PlatformCollection: InstanceCollection
```

Java

```
package com.intel.mobile.platform

public class PlatformCollection extends InstanceCollection
```

Processor*

The Processor namespace provides access to information, methods, and events related to system processors. The information model for processors is shown in Figure 9.

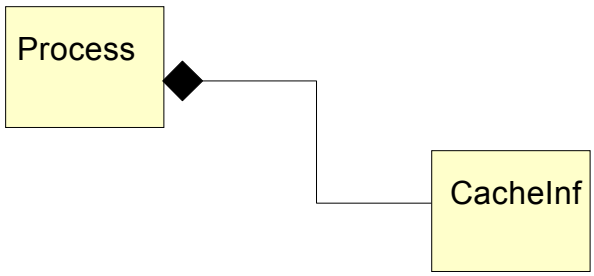

Figure 9. Processor model

ProcessorClass

The Class object for processors.

C++

```
namespace Intel::Mobile::Processor

class ProcessorClass: ClassObject
```

C#

```
namespace Intel.Mobile.Processor

public class ProcessorClass: ClassObject
```

Java

```
package com.intel.mobile.processor

public class ProcessorClass extends ClassObject
```

* Support for **Processor** on Windows CE has been valided on several PDAs including Dell* AXIM* x30 Handheld, Acer* n10 Handheld, Asus* MyPal A620, HP iPAQ Pocket PC h5500, HP iPAQ H1940, MiTAC MIO 558L Pocket PC, Symbol* MC9060 handset with English and Chinese versions of Windows Mobile 2003.

ProcessorInstance

The Instance object for processors.

C++

```
namespace Intel::Mobile::Processor

class ProcessorInstance: public InstanceObject
```

C#

```
namespace Intel.Mobile.Processor

public class ProcessorInstance: InstanceObject
```

Java

```
package com.intel.mobile.processor

public class ProcessorInstance extends InstanceObject
```

HTStatusType

The table below lists the possible statuses of Hyper-Threading Technology[1].

Value	Meaning
NotPresent	Hyper-Threading Technology not detected (not present).
DisabledByHardware	Hyper-Threading Technology not enabled (disabled by hardware).
DisabledBySoftware	Hyper-Threading Technology not available (disabled by software).
Enabled	Hyper-Threading Technology detected, enabled, and available.

[1] Hyper-Threading (HT) Technology requires a computer system with an Intel® Pentium® 4 processor supporting HT Technology and Hyper-Threading Technology-enabled chipset, BIOS and operating system. Performance will vary depending on the specific hardware and software you use. See www.intel.com/info/hyperthreading for more information including details on which processors support HT Technology.

Properties

Name	Type	Get	Set	Static	Description
Manufacturer	**String**	X		X	Processor manufacturer.
Brand	**String**	X		X	Processor brand.
ID	**String**	X		X	Unique identifier for the processor.
CurrentFrequency	**uint32**	X			Current processor frequency in MHz.
MaxFrequency	**uint32**	X		X	Maximum processor frequency in MHz.
NumberOfPhysical	**int32**	X		X	Number of physical processors.
NumberOfLogical	**int32**	X		X	Number of logical processors.
LogicalPerPhysical	**int32**	X		X	Number of logical processors per physical processor package.
HTSupported*	**Boolean**	X		X	Indicates whether Hyper-Threading Technology is supported and available for use. See the **HTStatus** property below for more detailed information.
HTStatus*	**HTStatusType**	X			Indicates the detailed status of Hyper-Threading Technology. See the **HTStatusType** table above for a list of values.
CPUIDSupported*	**Boolean**	X		X	Indicates whether the CPUID instruction is supported.
MMXSupported*	**Boolean**	X		X	Indicates whether Intel® Wireless MMX™ technology is supported.
SSESupported*	**Boolean**	X		X	Indicates whether SSE is supported.
SSE2Supported*	**Boolean**	X		X	Indicates whether Intel® Streaming SIMD 2 Extensions (SSE2) are supported.
SSE3Supported*	**Boolean**	X		X	Indicates whether Intel® Streaming SIMD 3 Extensions (SSE3) are supported.
DAZSupported*	**Boolean**	X		X	Indicates whether the Denormals Are Zero mode is supported.

* Only applicable on Windows XP Professional

Name	Type	Get	Set	Static	Description
RDTSCSupported*	**Boolean**	X		X	Indicates whether the Read Time Stamp Counter is supported.
CMOVSupported*	**Boolean**	X		X	Indicates whether the CMOV instruction is supported.
Type	**String**	X		X	Processor type.
Family*	**int32**	X		X	Processor family.
Model*	**int32**	X		X	Processor model.
Stepping	**int32**	X		X	Processor stepping version.
CacheLineFlushSize	**int32**	X		X	Processor cache line flush size.
DataWidth*	**uint32**	X		X	(bits) Processor data width.
Role*	**String**	X		X	Free-form string describing the role of the processor (e.g., central processor, math processor).
CurrentVoltage*	**uint32**	X			(tenth-volts) Current processor voltage.
MaxVoltage*	**uint32**	X		X	(tenth-volts) Maximum processor voltage.
Temperature*	**uint32**	X			(tenths of degrees centigrade) Current processor temperature.
FrequencyThrottlingSupported*	**Boolean**	X		X	Indicates whether the system supports processor frequency throttling.
ThermalControlSupported*	**Boolean**	X		X	Indicates whether thermal control is supported.

Methods

Return Value	Signature	Description
CacheInfoVector	**GetCacheInfo***	Returns a vector. The vector includes a **CacheInfo** for each type of cache associated with the processor. See the **CacheTypeValue** table below for a list of types of cache. To enumerate each element of the returned vector, the iterator should be created and set from the begin to end, like **vector** template in C++, **ArrayList** class in Common Language Runtime (CLR), and **List** class in Java.

* Only applicable on Windows XP Professional.

CacheInfo

The information model for processor caches is described below.

CacheTypeValue

The table below lists values for the types of cache.

Value	Meaning
L1Instruction	L1 instruction cache
L1Data	L1 data cache
L2Unified	L2 unified cache
Trace	Trace cache
L3Unifed	L3 unified cache

Properties

Name	Type	Get	Set	Static	Description
Type	**CacheTypeValue**	X		X	Indicates the type of cache. See the **CacheTypeValue** table above for a list of values. The following properties are associated with the cache type:
Size	**uint32**	X		X	(Kb) The size of the cache.
SetAssociative	**uint32**	X		X	(value) The type of the cache (for example, 4-way set associative.
LineSize	**uint32**	X		X	(Kb) The size of each line in the cache.
Lines	**uint32**	X		X	The number of lines the cache contains.

ProcessorCollection

The Collection object for Processor.

C++

```
namespace Intel::Mobile::Processor

class ProcessorCollection: public InstanceCollection
```

C#

```
namespace Intel.Mobile.Processor

public class ProcessorCollection: InstanceCollection
```

Java

```
package com.intel.mobile.processor

public class ProcessorCollection extends InstanceCollection
```

Appendices

References

[1] Huntsman, Justin B. *Mobile Application Architecture Guide*, Intel Corp., Dec 2003, www.intel.com/update/departments/software/sw12031.pdf

[2] *Electronic business using extensible markup language (ebXML) message service specification (MSS) v2.0*, Organization for the Advancement of Structured Information Standards (OASIS), http://www.ebxmlforum.org/

[3] *Web Services Reliability (WS-Reliability)*, Microsoft et al., http://otn.oracle.com/tech/webservices/htdocs/spec/ws-reliability.html

[4] *Web Services Reliable Messaging Protocol (WS-ReliableMessaging)*, Oracle et al., http://xml.coverpages.org/ni2004-03-11-a.html

[5] *WAP Binary XML Content Format,* Wireless Application Forum: World Wide Web Consortium, http://www.w3.org/TR/wbxml/

[6] *Common Information Model* (*CIM) v2.7*, Distributed Management Task Force, Inc. (DMTF), http://www.dmtf.org/standards/cim

[7] *Advanced Configuration And Power Interface Specification* Revision 2.0c, August 25, 2003, http://www.acpi.info/spec.htm

Glossary

Term	Description
Adaptive radio	A radio which, through software or other inputs, is able to adapt to the most appropriate signal. An adaptive radio is also able to adapt its own signal characteristics, such as modulation or output power.
Agile radio	A radio with the ability to acquire a signal from a wide variety of frequencies.
AP	Access point.
API	Application programming interface.
ARA	Adaptive radio architecture.
ATIM	Announcement traffic information message.
BCCH (GSM carrier)	Broadcast control channel, the BCCH is used to broadcast information, such as cell and network identity and operating characteristics of the cell (current channel structure, channel availability, and congestion).
BSS	Base station system.
BT	Bluetooth.
CCA	Clear channel assessment (in the context of detection schemes.

Term	Description
CDMA	Code-division multiple access.
CLR	Common language runtime.
Cognitive radio	A radio that acquires a space in the free spectrum in which to communicate by negotiating RF parameters with its peers or access points.
CPICH (received signal code power)	Common pilot channel, in the context of received signal code power (RSCP).
CS	Carrier sense.
CTS	Clear to send.
DES encryption	Data encryption standard. A simple encryption scheme.
Device stack	An operating system stack, such as the networking or USB stacks.
Discrete radio	Radio hardware that runs a known communications protocol.
DSSS	Direct-sequence spread spectrum. One of two types of spread spectrum radio, the other being frequency-hopping spread spectrum.
ED	Energy detect.
EHCC	Enhanced HCC.
FCS	Frame check sequence.
FHSS	Frequency-hopping spread spectrum, spectrum technology.
GPRS	General packet radio service.
GSM	Global system for mobile communications. Usually refers to the European standard GSM operating on 900 MHz and 1800 MHz bands, but in North America refers to the 1900 MHz band.
HCC	Honeycomb chain channel. A pattern-hopping algorithm.
HR	High-rate.
HRDSSS	High-rate direct-sequence spread spectrum.
IEEE	Institute of Electrical and Electronics Engineers.
IR	Infrared.
Legacy radio	A discrete device that whose driver is not ARA-aware.
LSB	Least significant bit.
MAC	Machine address code. This address is a device’s unique hardware number.

Term	Description
MDSU	MAC data service unit. The header, MSDU, and trailer contain information such as the address, IEEE 802.11-specific protocol information, information for setting the NAV, and frame check sequence for verifying the integrity of the frame.
MSB	Most significant bit.
NAV	Network allocation vector.
NDI	Network driver interface.
NDIS	Network Driver Interface Specification.
NIC	Network interface card.
OFDM	Orthogonal frequency division multiplexing.
OID	Object identifier used with NDIS.
OMA	Open Mobile Alliance. See www.openmobilealliance.org.
OSI	Open systems interconnection.
OTA	Over the air.
PBCC	Packet binary convolutional coding, a modulation implementation.
PCCPCH	Primary common control channel, in the context of received signal code power (RSCP).
PHY	Physical.
PIN	Personal identification number.
Protocol	The series of pipeline elements implementing a wireless functionality, such as 802.11a baseband or lower MAC functionality, running on a reconfigurable device.
PUK	PIN unblocking key.
RC2	A block encryption algorithm. The input and output block sizes are 64 bits each. The key size is variable, from one byte up to 128 bytes, although the current implementation uses eight bytes.
RC4	A stream cipher algorithm. RC4 is used for file encryption in products such as RSA SecurPC∗. It is also used for secure communications, as in the encryption of traffic to and from secure Web sites using SSL protocol.
RC5	A fast symmetric block cipher suitable for hardware or software implementations. A novel feature of RC5 is the heavy use of data-dependent rotations. RC5 has a variable word size, a variable number of rounds, and a variable-length secret key. The encryption and decryption algorithms are exceptionally simple.

∗ Other names and brands may be claimed as the property of others.

Term	Description
RCA	Reconfigurable communication architecture.
RCPI	Received channel power indication.
Reconfigurable radio	Radio hardware that can be configured at run time to run one or more different communications protocols.
RF	Radio frequency.
RIM	Radio information model defined by the ARA.
ROS	Radio operations subsystem.
RPI	Relative power index.
RSCP	Received signal code power.
RSSI	Received signal strength indication.
RTS	Ready to send.
Rx	Receive.
SCA	Software communication architecture.
SDR	Software defined radio.
SIM	Subscriber identification module. Applies to GSM networks.
SR	Software radio.
SSID	Service set identifier.
SSL	Secure sockets layer.
STA	Station.
TTM	Time to market.
TUs	Time units.
Tx	Transmit.
UMTS	Universal mobile telephone service.
USB	Universal serial bus.
WEP	Wired equivalent privacy protocol
WLAN	Wireless local area network
WPAN	Wireless personal area network
W-USB	Wireless USB
WWAN	Wireless wide area network